Neocognitron: Application to the Handwritten Hindi Numerals

by

Tambi Mohammad Sharief Baik

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

June, 1995

Neocognitron: Application to the Handwritten Hindi Numerals

BY

Tambi Mohammad Sharief Baik

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Science

June 1995

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by

Tambi Mohammad Sharief Baik

under the direction of his thesis advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee:

Dr. Meng Er (Chairman)

٩٥/٧٨٤

Dr. Muhammed S. Al – Mulhem (Member)

28/6/95

Dr. Mostafa Aref (Member)

Kanaan Faisal 28/06/95

Dr. Kanaan Faisal (Member)

٩٥/٧٨٤

Department Chairman

Dean, College of Graduate Studies

28.6.95

Date

Neocognitron: Application to the Handwritten Hindi Numerals

Tambi Mohammad Sharief Baik

Computer Science

June 1995

To my father and mother

whose support and prayers

led to this achievement.

To my dear sisters.

Acknowledgments

All praise be to Allah for his limitless help and guidance. Peace and blessings of Allah be upon his prophet Muhammad.

Acknowledgment is due to King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for the generous help and support for this research.

I would like to express my profound gratitude and appreciation to my thesis chairman, Dr. Meng Er, Professor of Information and Computer Science, for his guidance and patience throughout this thesis. I would also like to thank Dr. Muhammed Al-Mulhem, Assistant Professor and chairman of Information and Computer Science Department, Dr. Mostafa Aref, Assistant Professor of Information and Computer Science, and Dr. Kanaan Faisal, Assistant Professor of Information and Computer Science, for their consistent support and valuable suggestions. Special Thanks goes to Dr. Kamal Jambi, Assistant Professor of Computer Science Department of King Abdul Aziz University for providing some good research materials.

I also wish to thank faculty, research assistants, and the staff members of the Information and Computer Science Department. The encouragement and good wishes of my friends are also worthy of acknowledgment. Finally, special thanks must be given to my family for their encouragement and moral support.

Contents

List of Tables

List of Figures

Abstract

Name: Tambi Mohammad Sharief Baik

Title: Neocognitron: Application to the handwritten Hindi Numerals

Major Field: Computer Science

Date of Degree: June, 1995

Character recognition is a very important application in the field of computer vision. The Neocognitron is a multilayered neural network, originally designed by Fukushima, that has the capability of pattern recognition. Performance of the Neocognitron is not affected by translation, scaling, or deformation of the patterns. The objective of this thesis is to design the Neocognitron to recognize the handwritten Hindi numerals. A proper training set was designed that consisted of features to be recognized at every layer. In addition, detailed analyses were carried out to decide on proper parameter assignments. A recognition rate of 95% was achieved with a 5% error rate compared to a 16% error rate reported by Fukushima. The designed Neocognitron maintains excellent performance under noisy environment. At 3 SNR, the Neocognitron maintains a recognition rate of not less than 85%.

Master of Science Degree
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

خلاصة الرسالة

إسم الطالب الكامل : تامبي محمد شريف بك

عنوان الدراسة : تطبيق ال"Neocognitron" على الأرقام الهندية

التخصص : علوم الحاسب الآلي

تاريخ الشهادة : حزيران/ يونيو ١٩٩٥ م

يعتبر التعرف الآلي على الأحرف تطبيق مهم جدا في مجال بصريات الحاسب الآلي. ال"Neocognitron" هو عبارة عن شبكة عصبية متعددة الطبقات، صممها في البدء فوكوشيما، ولديها القدرة على تمييز الأنماط. مستوى أداء ال"Neocognitron" لا يتأثر بإزاحة، تغيير حجم، أو تغيير شكل الأنماط. الغرض من هذه الرسالة هو تصميم ال"Neocognitron" للتعرف على الأرقام الهندية المكتوبة بخط اليد. قمنا بتصميم مجموعة تدريب مناسبة مكونة من خصائص تستخلص في كل طبقة. بالإضافة إلى ذلك، قمنا بتأدية تحليلات مفصلة لتحديد قيم مناسبة لمحددات النظام. في النهاية تم التوصل إلى نسبة تعرف بحدود ٩٥٪ مع نسبة خطأ ٥٪ بالمقارنة مع نسبة خطأ ١٦٪ والتي توصل إليها فوكوشيما. ال"Neocognitron" المصمم يحافظ على مستوى ممتاز من الأداء حتى في بيئات عالية التشويش. حتى مع مستوى من التشويش يصل إلى ٣ SNR، يحافظ ال"Neocognitron" على نسبة تعرف أعلى من ٨٥٪.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول و المعادن
الظهران، المملكة العربية السعودية

حزيران/ يونيو ١٩٩٥ م

Chapter 1

Introduction

Character recognition plays a very important role in many applications. For example, in the United States, the mail service processes 550 million letters everyday. Out of these letters, 85% have machine printed addresses; and the rest are handwritten. Sorting these letters (which is based on their zip codes) costs $40/1000 letters when done by hand. This amount drops to $4/1000 letters when performed by an OCR. The OCR, working at a speed of 13 letters/sec, succeeds in sorting 50% of the machine printed envelopes. The goal is to increase this Figure to 90% for machine printed and 50% for handwritten envelopes [Bra93].

Other equally important applications include: data entry from bank checks, processing manually entered tax forms, handling office memos and magazine articles,

interpreting spreadsheets, and devising reading machines for the blind.

In this chapter we define the problem and the objective of our work along with the undertaken approach. In addition, we investigate the relation of the neocognitron to vision. We discuss this relation from physiological and psychological point of views.

1.1 Problem Definition

The neocognitron is a neural network that is designed by Fukushima. The basic idea behind the design of the neocognitron is to mimic the behavior and operation of the human visual system. Many concepts taken from the visual system are incorporated into the design of the neocognitron. Examples include:

Receptive Field: Every cell in the visual system has a receptive field. For example, in the retina, a bipolar cell has a receptive field that covers a group of receptor cells. In addition, a ganglion cell has a receptive field covering a group of bipolar cells. This concept is also apparent in the layered architecture of the visual cortex.

Excitation and Inhibition: Cells like the ganglion cells receive excitatory and inhibitory connections. Response of a cell depends on the aggregate input where excitation must exceed inhibition for the cell to respond. Cells like lat-

eral geniculate cells, simple and complex cells also behave in the same manner.

Simple and Complex Cells: The visual cortex contains two major kinds of cells, namely simple and complex. Simple cells are responsible for extracting features, while complex cells are responsible for correcting and blurring of features.

Hierarchy: Cells in the visual cortex seem to be organized in a hierarchical manner. Entry level layers detect local features, such as lines of different orientations. Higher level layers detect more global features and present more abstract description of the image.

Work on the neocognitron started nearly 15 years ago, with an implementation presented by Fukushima for the recognition of a set of arabic numerals [FM82]. Recognition of the complete set was presented in [FMi83] and [Fuk88b]. Recognition of the english alphabet was presented in [FW91]. The idea of *selective attention* is presented in [FI93].

The market is in full demand for arabic character recognition applications. This is due to the fast computerization that is ongoing. However, research on arabic character recognition is still in its early stages. With nearly one decade of reasearch, the field of arabic character recognition did not have any major achievement. Besides, nearly all of the work concentrated on the recognition of the arabic alphabet. The

Hindi numerals, which are the numerals we use in our arabic writing, did not have their share in the reasearch.

The objective of this thesis is to design a system for the recognition of the handwritten Hindi numerals. The targeted system should not be affected by translation or scaling of the numerals. In addition, it must tolerate minor deformations and minor rotations.

Our approach is based on the architecture of the neocognitron. Utilizing the neocognitron requires two major tasks:

- Designing a suitable training set that must be taught to the neocognitron
- Deciding on proper parameter assignments

Any character recognition application will be useless if it possesses a high error rate. Our system must have a very low error rate. However, since its targeted towards *handwritten* rather than typed or printed characters, an error rate of not more than 3% should be accepted.

1.2 Anatomy of the Visual System

The design of the neocognitron was first stimulated by discoveries of Dr. Hubel and Dr. Wiesel of Harvard university. After experimenting on the visual cortex of an anesthetized cat, they were able to detect two kinds of visual cortex cells, namely simple cells and complex cells [Hub72]. Simple cells were also discovered in monkeys [HW68] and in humans [Mar73].

The visual abilities of different creatures can not be judged by considering the eyes alone. In order to use the information gathered by the eye, it must be supplemented by a suitable nervous systems. It is because of differences in the nervous systems that animals with similar eyes structures differ in the amount and the way they perceive stimulus information [BD79].

In order to understand the mechanism of visual perception, one has to consider the anatomy of the visual system. In this section we try to shed a light on some major parts of the visual system.

1.2.1 Basic Structure of the Eye

In this description we start from the front of the eye imagining a light beam making its way through different parts of the eye. The first encountered part is the *cornea*

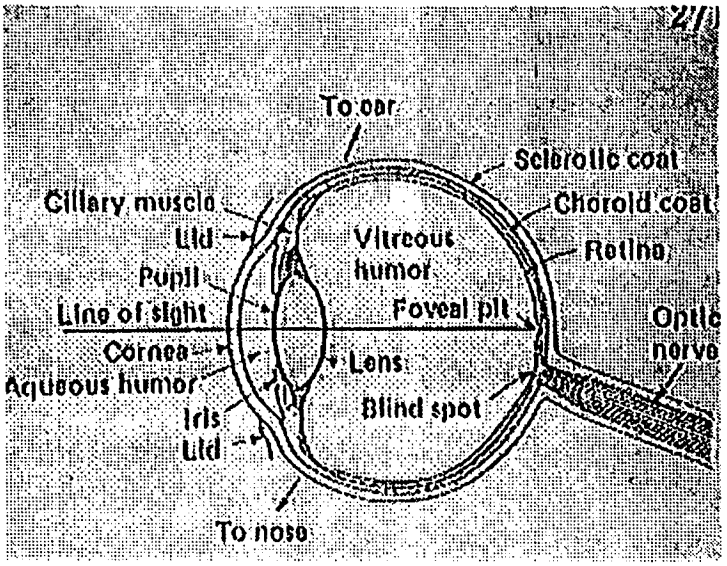


Figure 1.1: Basic structure of the eye. From [BD79]

which is a tough, thin, and transparent sheet of tissue blobbing out in a spherical fashion, see Figure 1.1.

Beyond the cornea, there is the *aqueous humor*, a watery fluid similar to the plasma or the fluid of the blood. It serves the purpose of passing nutritives and removing wastes from the *cornea* and the *lens*. Within the *aqueous humor* is the *iris*, a doughnut-shaped structure that gives the eyes their different colors. The hole in the center corresponds to the *pupil* which varies in size depending on the light level or varying nervous stimulations.

Immediately behind the *pupil* is the *lens*, a flexible, transparent, and filled with a gelatinous material. The *lens* plays a major role (along with the *cornea* and the *iris*)

in properly focusing incoming images via its ability to flatten and contract. These two jobs are accomplished by the *ligaments* and the *ciliary muscles.* The *ligaments* pull the lens into its flatter form. The *ciliary muscle* when contracting, releases the tension created by the *ligaments* and allow the lens to take a more rounded shape.

Behind the lens and occupying the space before the *retina* is a gelatinous substance called the *vitreous humor.* It is a passive transmitter for light. The *vitreous humor* passes nutritives to the *aqueous humor* to be passed to the *lens* and the *cornea.* In addition, it supports the *lens* and the *retina* [BD79].

1.2.2 Structure of the Retina

The retina is a network of blood vessels, neuron fibers and cell bodies, and receptor cells, see Figure 1.2. The retina holds up to 130,000,000 receptor cells. It is arranged in layers with blood vessels and long fibers at the front, followed by layers of neurons and fibers, and finally the receptors at the back. It comprises two important parts, namely: the *blind spot* and the *fovea* [BD79].

Blind spot : an area empty of receptors, and where blood and visual information pass through. Because of the absence of receptors in this area, parts of objects images projecting on it are not detected.

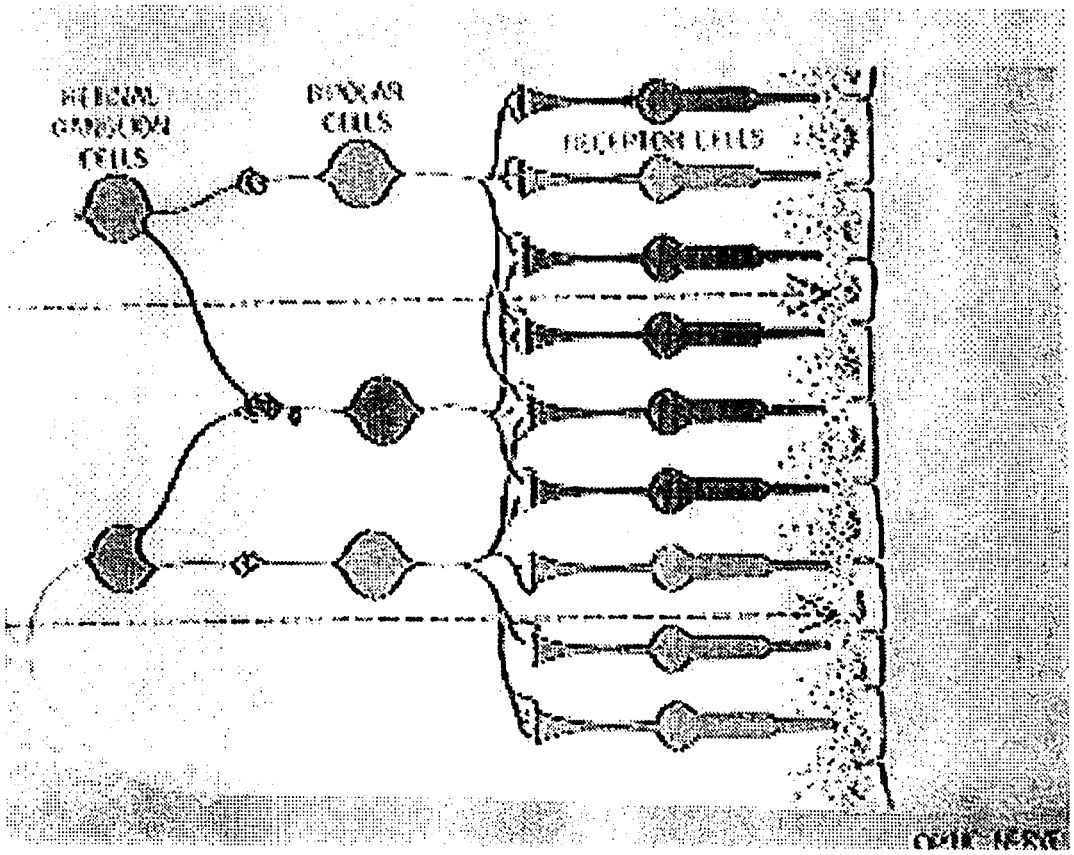


Figure 1.2: Structure of the retina. From [Hub72]

Fovea : a small depression at the center of the retina. The acuity of vision at the center of the retina can be explained by the fact that blood vessels and the neurons are pulled away to the side allowing light to strike the receptors with no obstacles. In addition, the receptors are thinner and denser.

Nerve cells transmit messages in the form of very short electrochemical impulses. Impulses of all nerve cells have the same amplitude, but they differ in frequency. The area of junction between nerve cells is called the synapse. Upon reaching the synapse, the impulse causes the release of a substance that diffuses to the next cell's membrane. There, this substance either excites the cell (making it fire) or inhibits it [Hub72].

The impulses travel starting at the receptor cells, through the bipolar cells to the ganglion cells. All of these cells exist in the retina. The ganglion cells then send their fibers to the visual cortex.

Every cell in the neocognitron has a receptive field. A set of contiguous cells in layer i sends its outputs to one cell in layer $i+1$. The idea of a receptive field stems from the fact that there is an M to N connections between the receptive cells and the bipolar cells, and between the bipolar cells and the ganglion cells. In this way, many ganglion cells are affected by one receptive cell. In addition, one ganglion cell is affected by many receptive cells, forming what is known as a receptive field

[Hub72].

Since the synapses might be either excitatory or inhibitory, one ganglion cell receives several excitatory and inhibitory connections and respond to the net effect of them. It was noted in [Kuf53] that the receptive field of a ganglion cell consists of "on" and "off" regions. Shining light on the "on" region would excite the cell, and shining light on the "off" region would inhibit the response of the cell. Shining light on both regions produces very weak and diffused response [Kuf53].

A ganglion cell has connections leading from around 130 receptors comprising the receptive field of this ganglion cell. The receptive field of a ganglion cell takes a circular shape encompassing a circular center. The center responds to light in a fashion opposed to the rest of the field [BD79].

There are two kinds of ganglion cells that are similar in structure but different in response. One kind will respond to the presence of light at its center and the absence of light at the periphery, thus the name "on" center / "off" periphery, see Figure 1.3. The other kind is stimulated by the presence of light at the periphery and the absence of light at the center naming it "off" center / "on" periphery.

Extensive filtering and abstraction of information have taken place while transmitting information from receptive cells to the ganglion cells. With 130,000,000 receptive cells and 800,000 to 1000,000 ganglion cells, a lot of information reduc-

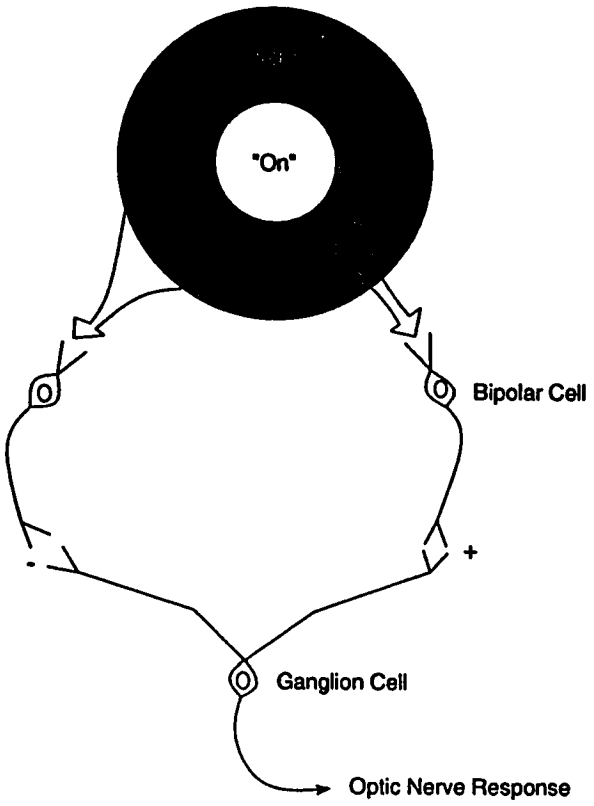


Figure 1.3: A ganglion cell and its receptive field. From [BD79]

tion and coding took place. This information reduction and coding is carried out by the lateral connections between retinal cells [BD79].

An optic nerve fiber upon leaving the ganglion cells terminates at the *lateral geniculate cells*[1]. These cells behave more or less as the ganglion cells. However, they have the function of increasing the disparity already present in the ganglion cells "on" and "off" responses [Hub72].

1.2.3 The Visual Cortex

The structure of the visual cortex is so complex that many specializations attempt to comprehend different facets of it. Disciplines such as perceptual and cognitive psychology, neurophysiology, neurology, neuroanatomy, embryology, neuropharmacology, mathematics, engineering, information theory, neural network theory, physics and many others are needed to study the working of the visual cortex [Jul91].

Cells at the visual cortex exhibit a layered structure with rich connections between layers. The incoming connections of the visual cortex lead from the *lateral geniculate body*, see Figure 1.4. Cells of the visual cortex accept inputs through receptive fields much the same way the lateral geniculate cells accept inputs [Hub72].

[1]Which is a midway body before the visual cortex

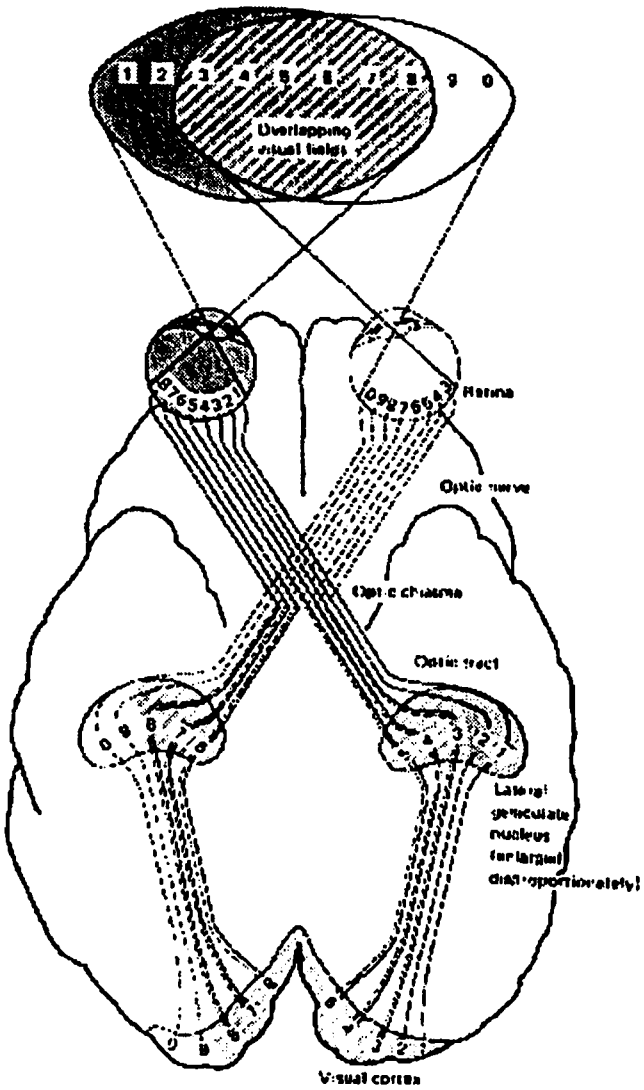


Figure 1.4: The visual cortex. From [BD79]

The two scientists, Hubel and Wiesel, carried a very important experiment in order to unveil some properties of the visual cortex. In their experiment, Hubel and wiesel placed a wide screen 1.5 meters away from an anesthetized cat. Several patterns were shone on the screen. At the same time, fine microelectrodes penetrated the visual cortex. The responses of these electrodes were recorded for every pattern. The presented patterns consisted of lines and edges of different orientations [Hub72]. They were able to classify the cortical cells into three major types:

Simple Cells: It has been ascertained that these cells are present in cats [Hub72], in monkeys [HW68], and in humans [Mar73]. Simple cells have receptive fields that are more complex than those of the *lateral geniculate cells.* It was shown that a particular receptive field of a simple cell consists of a long slit of "on" region surrounded by "off" region. The maximum response occurs when all cells of the "on" region are excited and all cells of the "off" region are inhibited, corresponding to a particular line orientation.

A simple cell has a receptive field covering a group of lateral geniculate cells. Each lateral geniculate cell in turn has a receptive field that covers a group of ganglion cells. The concept of *excitation* and *inhibition* in the cortical simple cells is the basis for formulating the response of the simple cells in the neocognitron.

Complex Cells : Complex cells get their inputs from simple cells. They also have

tendency to respond to specific line orientations. However, when the stimulus line is in motion, the complex cell differs from the simple cell in that its response is sustained. The firing continues as the line moves until it crosses the border of the receptive field. However, lines moving in the opposite direction elicit no response from this complex cell [Hub72].

Hypercomplex Cells : The highest order of cortical neurons studied by Hubel and Wiesel is the hypercomplex cell. It is responsible for detection of moving lines of a particular orientation and of a particular length. A receptive field of a hypercomplex cell extends to cover a group of complex cells.

As [Gre74] describes it, the discovery of Hubel and Wiesel implies that objects are not pictured, but rather they are described in terms of features ranging from very local to global.

Hubel and Wiesel found that deeper layers of the visual pathway are responsible for the representation of more abstract features of the retinal patterns. Up through the hierarchy of layers, more and more information seems to be consolidated to form a more general representation of the patterns [Gre74].

The layered architecture of the neocognitron mimics that of the visual pathway. Every layer in the neocognitron consists of two sublayers : S-sublayer and C-sublayer. S-cells detect specific features and C-cells blurs the outputs of preceding S-cells to

produce an invariant response. Layers at a lower level or the entry layers of the neocognitron detect local features of the input patterns. Higher level layers or exit layers detect more global features. Features detected at layer i are combined to form a more global feature to be detected at layer $i + 1$.

It is believed that there are two principal types of visual systems. The first type is exemplified in the frog, the rabbit, the ground squirrel, and the pigeon. In this visual system, the retina is a complex structure. Stimulus such as edges, color, contrast, orientation, and directional movements are processed intensively within the retina. The second type of visual systems is the one present in cats, monkeys, and humans. In this type, the ganglion cells represent the highest level of complexity in the retina. They are responsible for detecting simultaneous contrast between centers and peripheries of their receptive fields. In some cases, they are also concerned with color information. Features like edges, oriented slits of light, or directional movement are dealt with only at the visual cortex [Mic72].

The above discussion focused on seven types of cells that pertain to the visual perception system. These cells are organized into a layered architecture where every layer gets its input from the preceding one. Higher level layers offer more precise and informative description of the input patterns than lower level layers. Table 1.1 presents a list of these cells, sorted by increasing complexity, along with different parts of the human visual system containing them.

Cell type	Location
Receptor	Retina
Bipolar	Retina
Ganglion	Retina
Lateral Geniculate	Lateral Geniculate Nucleus
Simple	Visual Cortex
Complex	Visual Cortex
Hypercomplex	Visual Cortex

Table 1.1: Ordered list of complexity of the first seven cell types of the visual system

1.3 Introduction to Visual Perception

The design of the neocognitron, although triggered purely by the anatomy of the visual system, has a supporting psychological background. In perceptual processing, many scientists believe that one task of primary perception is detection of features [BD79]. A set of unique features characterizing letters of the english alphabet were postulated by Gibson [Gib69]. Some categories of these features seem to fit into the design of the neocognitron.

Studies of the first stages of the central nervous system of the monkey suggests a similarity between the visual system of the monkey and the human's visual system. The first retinal and cortical stages are called "bottom-up" stages as opposed to higher cortical stages of semantic memory and symbolic processing which are called "top-down" stages. The "bottom-up" stages are also referred to as early vision [Jul91]. While this is the accepted view, computational vision is sometimes thought

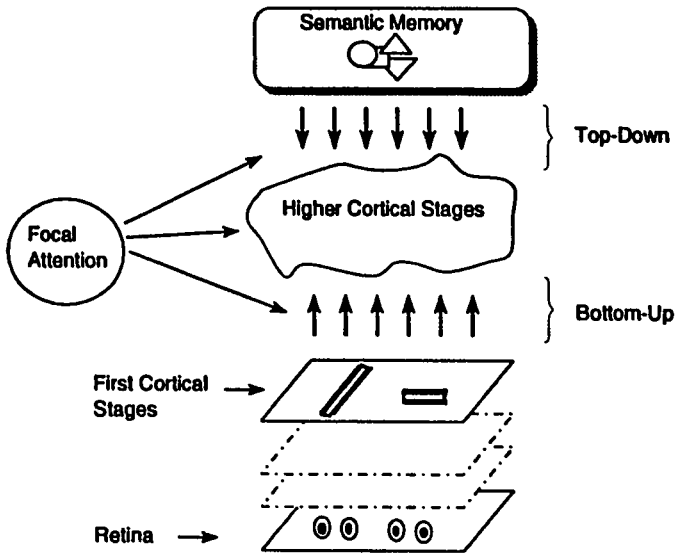


Figure 1.5: Bottom-up vs top-down processes in the visual system and focal attention. From [Jul91]

of as purely "bottom-up" [Mar82].

Figure 1.5 depicts the "bottom-up" and "top-down" views of the nervous system. Although the Figure suggests a layered information stream, each of "bottom-up" and "top-down" stages has subprocesses with feedback loops. In addition "bottom-up" and "top-down" processes are elaborately interconnected.

Focal attention is a "top-down" process that permits the observer to analyze different areas of the visual field with out eye movements. It also allows for inspection of different cortical areas. It is a mechanism that can scrutinize many processing stages. However, it can not be drawn to the earlier perceptual stages as well as most of the higher perceptual stages [Jul91].

Now, we turn to the discussion of major aspects of perceptual processing. Perceptual processing involves two main stages, namely: *primary perception* and *secondary perception* [BD79]. Other scientists refer to these processes by different names. For example, They are called *preliminary sensory processing* and *pattern recognition* in [DB73]. They are also referred to as *preattentive processing* and *focal attentive processing* in [Nei76].

Primary perception is responsible for feature extraction. Features at this level include information relating to contrast, location, length, slant, and motion. Specific figure shapes do not seem to be determined at this level. Primary perception operates in parallel across the visual field. The outcome of this stage is a crude and global model representing objects as collections of lightness contrasts.

Secondary perception receives information produced by the primary perception and applies *differentiation* and *enrichment*. Differentiation process ensures that the percept is adequately consistent with the world. Enrichment process gets information about the percept and compares it with information from long term memory in an effort to validate the consistency of the percept with what the brain already knows about the world. Discrepancies are solved in favor of current information.

However, in [Gib66] Gibson takes a somewhat different view when considering secondary perception. He believes that the mind relies heavily on the differentiation

process instead of the enrichment process. He postulates that the mind keeps on differentiating until it achieves clarity of perception. For the perception of objects, Gibson's differentiation theory implies the detection of distinctive features to form an abstraction of the general properties.

In his theory of information pickup, Gibson also asserts the adjustment of organs during the course of training that facilitates an easier information gathering and processing. Organ adjustments are controlled by lower centers of the nervous system as opposed to conceptual attention which is controlled by the highest centers.

Training affects the secondary perception only involving development of more effective attentional eye movement and changes in the way successive foveal glimpses are remembered and integrated.

Since light changes and light tracking exist in infants right after birth, it suggests that primary perception processes are hooked up in the brain and that learning seems to have little influence on their development [Hoc68].

A set of defining features for the english alphabet were postulated in [Gib69]. Every letter can be described in terms of the following features:

Lines: horizontal,vertical,diagonal with slope 1, or diagonal with slope -1.

Curve: closed as in 'B' and 'D', open vertical as in 'J' and 'U', or open horizontal

as in 'C' and 'G'.

Intersection: as in 'A', 'E', 'F', and 'H'.

Redundancy: presence of cyclic change as in 'E', 'M', and 'W' , or presence of symmetry as in 'A', 'H', 'I', and 'T'.

Discontinuity: vertical as in 'A', 'F', 'T', and 'X', or horizontal as in 'E', 'F', and 'Z'.

Characters sharing several features are apt to be confused. Each feature detector examines input from primary perception for the presence or absence of this feature. These feature detectors need to develop over time. Indeed, human beings take few years after birth to be able to differentiate between characters [Gib69].

Chapter 2

Literature Review

In our daily life, we are faced by countless patterns. These patterns can be classified into two main groups: abstract and concrete. Abstract patterns include ideas and arguments and recognition of these patterns is termed *conceptual recognition.* The other class of patterns include symbols, pictures, characters, speech waveforms, 3D objects, and many others [Bow92]. Within the context of this thesis the term *pattern recognition* refers to concrete patterns.

Pattern Recognition involves many applications. Some of the examples in [Sch92] are:

- Image preprocessing, segmentation, and analysis
- Computer vision
- Seismic analysis

- Radar signal classification
- Face recognition
- Speech recognition
- Fingerprint identification
- Character recognition
- Medical diagnosis
- Weather forecasting

Pattern recognition is generally defined as "the science that concerns the description or classification of measurements." It involves three major approaches that are interrelated in practice:

- Statistical Pattern Recognition
- Syntactic Pattern Recognition
- Neural Pattern Recognition

In the rest of this chapter, a discussion of these major approaches will be presented. In the course of this discussion, examples of arabic character recognition systems will be introduced. In some instances, examples of non-arabic systems will be incorporated for the sake of completion of discussion. A good survery of different arabic character recognition approaches is presented in [Jam91].

2.1 Statistical Pattern Recognition (StatPR)

Statistical pattern recognition assumes a statistical basis for the the classification problem. Every class of patterns is characterized by the presence or absence of particular features. In addition, measurements of the quantified features are used to map every pattern into its associated class [Sch92].

Normally, features are represented by d-dimensional *feature vectors* constituting the feature space. Every value in this vector corresponds to a position on one of the axes of the feature space. The space is partitioned into subspaces, each corresponding to a particular pattern class. During the recognition phase, the system computes the different features' measures and finds out the position of the unknown form in the feature space, thus detecting its class [DH73]. Land type determination using satellite-based sensors is one application where StatPr is well suited.

One of the earliest approaches toward arabic character recognition is the work of Amin in [AKHM80]. An on-line system employing a graphic tablet is used for entering the characters. The system deals with isolated characters only. A statistical approach is used that considers the following parameters:

Total number of strokes : A stroke is defined as a continuous portion of the character that is written before lifting up the pen. A main stroke is defined as the

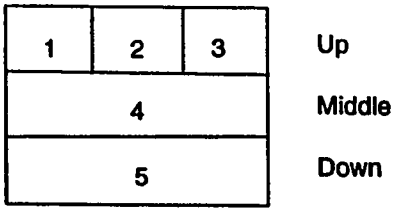


Figure 2.1: Position of a group of dots relative to the main stroke. From [AKHM80]

longest stroke. All isolated characters can have one or two strokes only.

Characteristic of groups of dots: These include

- number of dots (NBP) : A character can have up to three dots.
- position of a group of dots : It is assigned an integer value depending on the position of the group of dots relative to the main stroke, as shown in Figure 2.1

Form code of the main stroke: The four main segment directions are used to code the main stroke as exemplified in Figure 2.2

Presence of a zigzag

These features are used to classify the character into one of the character classes using a minimum distance classification.

In an off-line statistical approach reported in [AMSH89], different features are used to compose a "Code_book". In their implementation each vertical column of

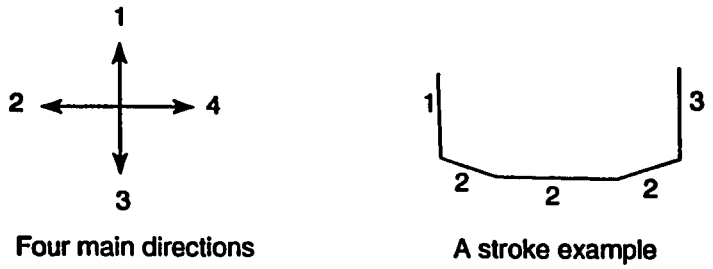


Figure 2.2: Coding of the main stroke. From [AKHM80]

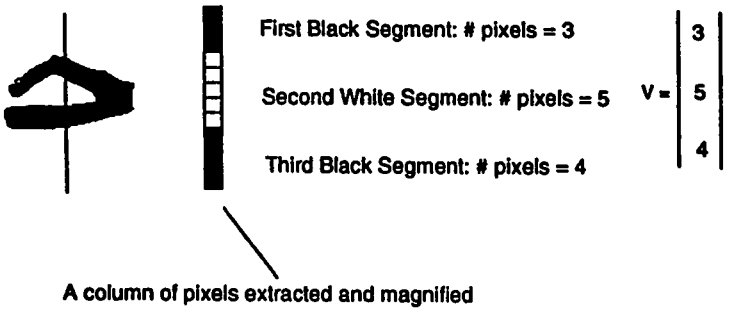


Figure 2.3: Feature extraction using column information. From [AMSH89]

pixels in the character is encoded in a vector. Number of pixels of alternating black and white segments along the vertical column are entered in the feature vector as shown in Figure 2.3.

As can be inferred from Figure 2.3 feature vectors can vary in dimensions and they are clustered into five groups containing 1,3,5,7, or 9 segments. During the

unsupervised phase of the learning, characters are presented sequentially and the feature vectors for every column are calculated. A decision is made whether to include the feature vector in the Code_Book or not. If the distance between the feature vector and a prototype vector is less than a threshold, the feature vector is added to its associated group. Otherwise, it is ignored. In the supervised phase, samples of characters are used to generate the conditional probability distributions of the elements of the Code_Book given each character class *a-priori probabilities.* The class probability distributions (*a-posteriori probabilities*) given a particular element in the Code_Book is then computed using *Baye's Rule.*

In the recognition phase, a sequential test is carried out on the cursive word from right to left. Each column of the word is matched with one of the elements of the Code_Book. A label is associated with this column. Using the position of the column within the word along with the label, the a-posteriori probabilities are retrieved . An accumulated probability distribution is computed at each step, and after satisfying a measure of uncertainty, recognition is achieved.

The concept of *moments* is well known in the field of classical mechanics. The two dimensional $(p+q)$th order moments of a density distribution function $p(x,y)$ are defined as

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q p(x,y)dxdy,\ p,q = 0,1,2,....$$

The use of moments in pattern recognition was stemmed by the introduction of *invariant moments*. Hu was first to formulate the definitions of invariant moments [Hu62]. These moments are invariant to translation, scaling, or rotation.

A statistical approach that depends on the *moments* is presented in [AYU92]. In this approach, words are segmented into characters. As a result, a character can have four different shapes depending on its position within the word. After segmenting the word into characters, a lower level segmentation is applied to isolate dots and zigzags (secondaries). This step follows the convention of decomposing an arabic character into a main part and a secondary part.

Secondaries are extracted during the segmentation step. An empty horizontal line indicates the presence of segments. The segment occupying more than half the number of black pixels is considered a primary part. Other segments are secondaries. A secondary can be one dot, two dots, three dots, or a zigzag. Secondaries are evaluated according to the algorithm presented in Figure 2.4

Normalized moments of vertical and horizontal projections are used to extract the following nine features:

- measures of flatness of the distribution (KV and KH)
- measures of skewness which are measures of the asymmetry of the distribution

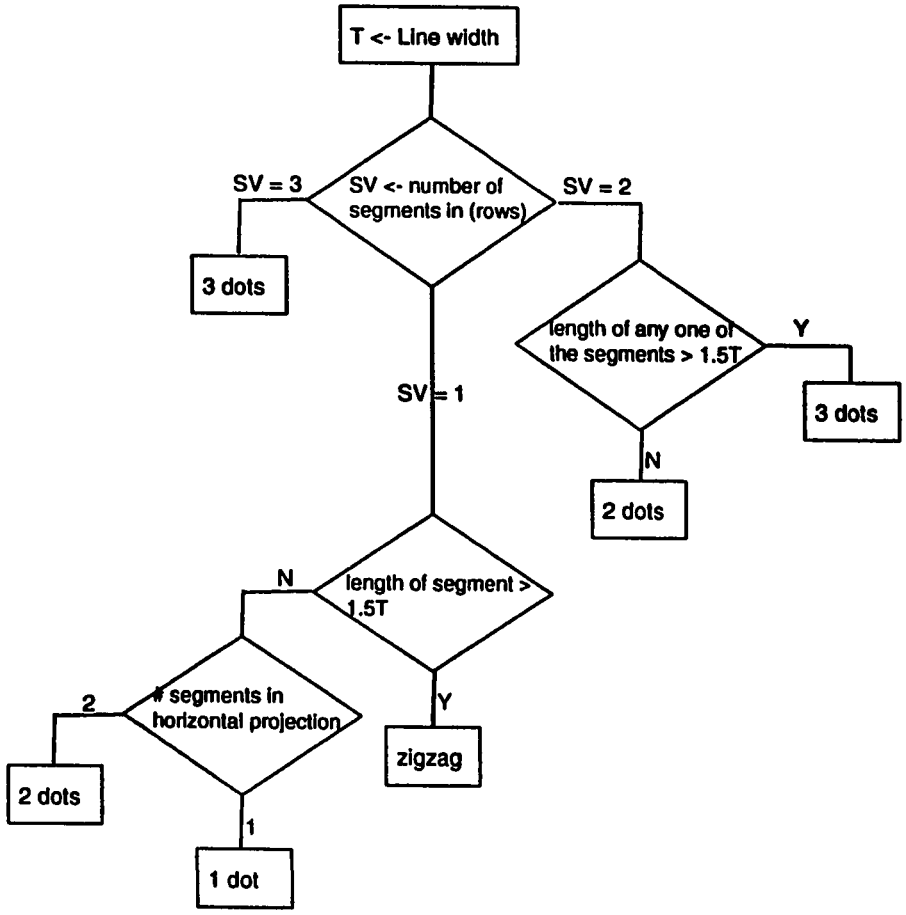


Figure 2.4: Algorithm to evaluate secondaries. From [AYU92]

(SV and SH)

- measures of normalized skewness and flatness (NV and NH)
- measures that relate the moments of the vertical and the horizontal projections for the same character (LW and VR and VV)

The terms V and H represent the measures derived from both the vertical and the horizontal projections.

These nine features together constitute a feature vector that completely characterize the main part of the character. Classification is accomplished using a quadratic Bayesian classifier. The classifier assumes that the prior probabilities of all classes are equal. The following term represents a measure of the distance between the vector x and a class with a mean m_i and covariance matrix $\sum_i$:

$$D_i^2 = (x - m_i)^T (\sum_i)^{-1} (x - m_i) + ln|\sum_i|$$

The vector x is classified into the group having the minimum value of D_i^2.

Another system that uses the concept of moments is reported in [EDRK90]. Invariant moments are used to characterize a segmented character. During training, 30 samples of the same character were used to calculate the mean and variance of a specific moment zone (allowable tolerance). An experiment was conducted to decide on the best number of moments to use. The number of moments was varied

between five and eleven and both the recognition rate and the recognition speed were calculated. Results are summarized in table 2.1

Number of Moments	Recognition Rate %	Recognition Speed (Char/Minute)
11	94	10.6
10	82	8.8
9	82	10.5
8	76	8.6
7	76	9.7
6	50	11.1
5	below 50	

Table 2.1: Performance vs number of moments. From [EDRK90]

It was found that an increase in the number of moments does not lead to an increase in the recognition speed. This is a result of the invocation of the recognition positioning and backward recognition and the excessive search for a match. Making use of the eleven moments seems to be the right choice considering both the recognition speed and the recognition rate.

Contour based features are utilized in [Mar92] in a statistical distance classifier. A contour is partitioned into segments and four main features of every segment are used in the classifier. In another approach reported in [Mah94], Fourier descriptors, curvatures features, and dots and holes features are combined to help in the recognition of arabic characters.

2.2 Syntactic Pattern Recognition (SyntPR)

In many applications, the segregation between patterns does not depend on the measurements of some features. Instead, the interrelations between features are used to abstract general structural pieces of information. In this approach a hierarchical description of a complex pattern is drawn from primitives or subpatterns [Fu74].

King-sun Fu was a leading Figure in the field of pattern recognition. He was the first to develop a complete and coherent theory of syntactic pattern recognition. His theory includes the use of stochastic grammars and languages to handle ambiguities and uncertainties in the pattern. His book [Fu86] presents a comprehensive description of pattern recognition theories and fundamentals.

Syntactic pattern recognition consists of two parts: *analysis* and *recognition.* The analysis part consists of primitive selection and grammatical inference. The recognition part consists of preprocessing, segmentation or decomposition, primitive recognition, and syntax analysis [Fu82].

SyntPR surpasses StatPr in many areas such as model-based vision and AI. Speech and character recognition are examples of applications where SyntPR excels. For example, in handwritten character recognition, a character comprises a set of features. In many cases, exact measurements of these features is not what is required

for proper classification. Instead, the relative ordering of these features that will lead to a proper structural representation is the major factor.

In [NST87] an optimum set of primitives is established. An initial set of primitives is established by measuring the correlation of every character with the rest of the characters. This set consists of the common shapes of the characters excluding dots. The procedure is repeated until the optimum set is reached where the decision is based on an aggregate correlation threshold. The correlation requires position by position matching of every feature with every character which is a very costly operation.

Number of strokes of a character is used to classify a character into one of four main groups [ESET90]. These groups include characters that consist of one, two, three, or four strokes. A stroke is either main or secondary. Secondary strokes include diacritics, zigzags, vertical lines, or diagonal lines. A tree structure is built for the classification of characters for every group. Classification is accomplished using primitives that include

- the ratio between width and height of the stroke or its inverse
- the number and sequence of minima and/or maxima in the X and/or Y directions of the main stroke

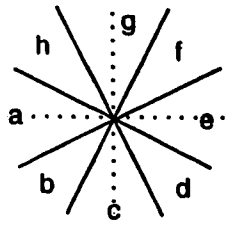
- the relative distance between the last point and the last y_{min}
- the relative horizontal and vertical distance between first and last points
- the position of the secondary stroke and other features

The drawback of this approach is that there are no general rules that would help in an easy and robust classification. Instead, every hierarchical classifier of every group employs a large number of heuristics and conditions. As a result, classifying a character involves satisfying many constraints. In addition, the number of conditions to be satisfied differs from a character to another resulting in a varying recognition time.

The approach of [GU89] relies on segmenting the word directly into strokes. A stroke finder algorithm finds the start and end points of a stroke. This is done by running a window of size 3 × 3 and identifying feature points. Strokes are either primary or secondary. A stroke after extraction is smoothed and sampled using *angular segmentation* algorithm.

After sampling, a stroke segment is classified into one of ten primitives using the eight directional codes shown in Figure 2.5. A primary classifier uses the primitives extracted along with the dot number, dot position, and the existence of a loop to give a decision on the character.

Code sequence	Primitive
c	
cba	
abc	
cbahg	
edcba	
abcde	
gfedc	
abcdedcba	
edcbabcde	
ghabcbahg	



Eight direction code

Figure 2.5: Primitives with associated code sequence. From [GU89]

In the approach presented in [EWS89] primitives are divided into two groups: stable and variable. A secondary stroke is defined as the longest continuous portion of the character which is written after the main stroke. This definition includes small segments of lines and zigzags, but it does not include the dots.

Secondary strokes and dots are considered as stable primitives, whereas the structure of the main stroke is the most variant part. The main stroke is coded using the directional primitives as shown in Figure 2.6. The stable features are coded using a four elements feature vector where the elements of the feature vector represent

Number of dots: zero, one, two, or three dots

Relative position of the dots: Above, within, below, or undecided (within or above) the character

Number of secondary strokes: zero or one

Slope of secondary stroke

A three stage classifier is used. In the first stage, the feature vector of the stable features is computed. This allows the character to be classified into one of ten groups. In the second phase, the string representation of the main stroke is matched against a prototype using a minimum distance classification to determine the associated subclass. As a final stage, a K-nearest neighbor recognition rule is

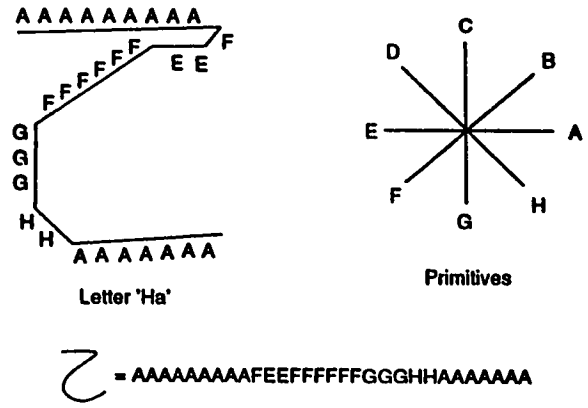


Figure 2.6: Representation of the main stroke. From [EWS89]

used to recognize the unknown pattern from the characters having the same matched prototype.

2.3 Neural Pattern Recognition (NeurPR)

Advances in physiology and in particular knowledge of the working of the nervous system and its interrelations with other systems have led the way for the emergence of a new means of storing and handling information.

Neural networks constitute a new paradigm that tries to mimic working of the biological systems [Ros59]. They are mostly appropriate for pattern recognition applications. Examples of neural networks are:

- Hopfield net

- Hamming net
- Carpenter-Grossberg classifier
- Multilayer perceptron
- Kohonen's self-organization feature maps

In [CBD+89] a neural network is implemented for the recognition of zipcodes. The network uses backpropagation for training. The segmentation process outputs characters that are 40×60 in size. Characters are then normalized to 16×16 using linear transformation. The designed network is not fully connected. Rather, it has a structured architecture sharing many properties with the neocognitron. Planes in the network are called *feature maps*. All neurons pertaining to the same plane share the same set of weights. The set of weights or connections form the receptive field. Every feature map is responsible for detecting a specific feature. The network consists of five layers. Layer 1 detects the lowest level features. Layer 2 averages and subsamples the results obtained by layer 1 (mimicking S and C layers in the neocognitron.) Layers 3 and 4 function similarly to layers 1 and 2 but they abstract more general features. The last layer is fully connected to layer 4. The performance was tested on samples provided by the U.S. Postal Service. The network has a 92% recognition rate with 1% substitution and 7% rejection.

Another implementation of a neural network uses *Gabor functions*. Gabor es-

tablished the optimality of these functions in maximizing the joint resolution in the spatial frequency domain in 1946. It was discovered that Gabor functions have resemblance to striate cortical cells profiles in 1980. They act as spatially localized feature extractors and they provide for a compact representation of the original image. This triggered the extensive use of Gabor functions in the computer vision field and in particular in pattern recognition applications. The neural network is trained on the coefficients of Gabor functions and the network after completion of training acquires a performance of 98.2% when tested on printed characters [Sri92].

Utilizing Kohonen's unsupervised learning is another approach reported in [HC93]. A three layered neural network is implemented. The first layer is trained to discriminate lines of four different orientations. When a feature is presented, the distance between the input and each of the output nodes is computed. An output node with the minimum distance will have its weights updated. Each layer of the network is trained on these features.

Next, the network is trained on complete handwritten characters. After the unsupervised training is finished, some different characters might still be classified as belonging to the same class. Supervised training is performed to correct for the misclassification. It is intended to examine the small discriminating differences between the characters and amplify these differences affecting the weights of the output layer. This is done by forwarding input to the output layer without passing through

intermediate layers. The network when trained on handwritten arabic numerals had a performance of 93% recognition accuracy [HC93].

This idea is reminiscent to that of *multiresolution* where lower resolution features are used for rough clustering and higher resolution features are used for fine classification.

Other approaches using neural networks include the boolean net algorithm [HK94] and a modified *backpropagation* neural network for invariant pattern recognition [YO93].

2.4 Other Approaches

2.4.1 Template Matching

One of the oldest techniques in pattern recognition is template matching. Template matching requires the convolution of the pattern with the template. This involves measuring the correlation between the two at every position and finding the maximum among all. For a pattern of $N \times N$ size, and a template of size $W \times W$, the computation time is of order $O(N^2W^2)$ [EB90].

In addition, template matching has other limitations. For example, two pixels

having different values, will have a difference of 1. This is very extreme keeping in mind that in character recognition, the absolute positions of pixels in an image are irrelevant to its recognition. As a result of this drawback, template matching techniques are oversensitive to shifts in position and distortions in shape of the stimulus patterns. It is necessary to normalize the position and the scale of the pattern before applying this technique. It is reported that template matching has a performance of 80% [Sri92].

A technique that enhances the performance of template matching is *fuzzy template matching.* The key idea of this technique is to distinguish between different degrees of mismatch. In fuzzy template matching, the difference of two pixels depeneds on their neighbors. The fuzzy template matching technique is reported to have a performance of 89% [Sri92].

2.4.2 Fuzzy Attribute Representation

In the system proposed by [CL94], a fuzzy structural recognition system of Chinese characters is used. An on-line graphic tablet is used to input the character where temporal information of character strokes is recorded. During training, characters are entered into a reference dictionary forming template characters.

A Chinese character is composed of individual strokes arranged according to

structural rules. To be more exact, a Chinese character contains more than one radical, where a radical is composed of strokes arranged according to structural relations. Structural relations between strokes of the same radical are stable, while positions and sizes of different radicals differ. Structural relationships can be one of the following: L-shape, T-shape, or cross-shape.

Since the program is supposed to handle handwritten characters, deformations of strokes are expected. Those deformations make identification of structural relationships between stroke segments fuzzy. This is why a fuzzy approach is used.

Initially preprocessing is conducted where every character undergoes normalization, smoothing, and extraction of two level line segments. Then, number of strokes and stroke segments are used as the criteria for preliminary classification.

In the recognition phase, each input handwritten character is represented by a fuzzy attribute graph. Every vertex in the graph stands for a character primitive. Each arc describes the spatial relationship between two stroke segments. These relationships include the relative positioning of the two stroke segments as well as information about stroke intersections.

Preliminary classification is used to form a candidate set of template characters. In order to check the similarity between the input character and a template, their two attribute graphs are matched. The template character with the maximum similarity

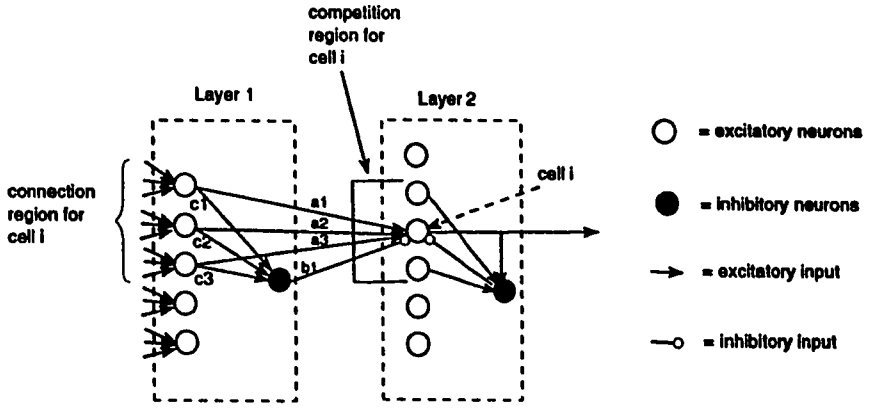


Figure 2.7: Cognitron layers. From [Woz88]

is decided to be the target character.

2.4.3 Cognitron

The cognitron is an earlier version of the neocognitron. Much of the work done on the neocognitron is based on this earlier model. The cognitron is a multi-layered hierarchical neural network. It has the ability of pattern recognition. In a given region of a layer, only the neuron that fires most strongly, has its presynaptic connections reinforced. In contemporary terms, we call this method competitive training. Figure 2.7 details the basic structure of the cognitron.

Inputs to neurons are either excitatory or inhibitory, as shown in Figure 2.7. The output of a neuron is determined by the ratio of the excitatory inputs to the inhibitory inputs. The cognitron does not have the ability to recognize shifted

patterns, nor can it detect deformed patterns.

In the next chapter the neocognitron model will be introduced. All necessary theoretical issues will be discussed in full detail.

Chapter 3

Theoretical Issues

The neocognitron is a multi-layered hierarchical neural network capable of pattern recognition, shown in Figure 3.1. The neocognitron can be trained in a supervised or unsupervised manner. The neocognitron acquires classification power quickly. Only few examples of the same pattern are necessary for the neocognitron to learn the pattern.

The neocognitron processes information in stages. The first layer of the neocog-

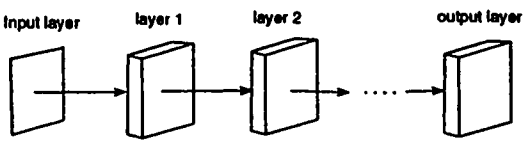


Figure 3.1: Neocognitron layer structure

nitron detects line segments and edges. The second layer detects more complex features including corners and crosses. The deepest layer of the network detects entire patterns serving as the recognition layer.

The neocognitron recognizes transformed patterns effectively. A pattern exposed to translation, deformation of shape, or scaling can still be recognized by the neocognitron. There is no need for a preprocessing stage to normalize the position, the scale or the deformation of the patterns. The only preprocessing step that is needed is to segment the collection of patterns and to present them in sequence to the neocognitron.

3.1 Structure and Behavior of the Neocognitron

The neocognitron consists of a cascade of few layers preceded by an input layer (U0), as shown in Figure 3.2. The input layer consists of photoreceptor cells. The image of the pattern is projected on the input layer. Every cell in the input layer corresponds to one pixel of the image [FM82].

Every layer of the neocognitron (except the input layer) consists of two sublayers, namely the U_S and the U_C sublayers. The U_C sublayer receives input from the preceding U_S sublayer. Cells of U_S are called simple cells. They serve as feature

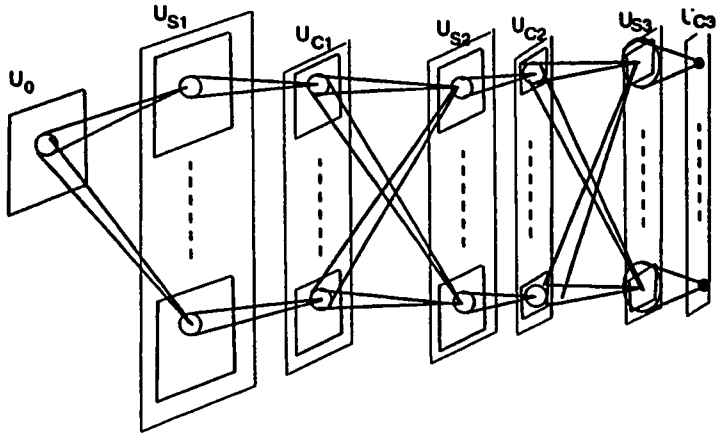


Figure 3.2: Illustration of the interconnections between layers. From [FM82]

extractors. Cells of U_C are called complex cells and they have the role of error toleration.

3.1.1 Role of Cell Planes

Cells in each sublayer are sorted into subgroups called cell planes. Every cell plane is responsible for extracting a particular feature from the input pattern. All cells in one cell plane share the same spatial input connections. Every cell detects the same feature but from a slightly different position.

Cells in the first layer extract local features such as lines of particular orientations. Cells in the second layer extract more global features such as parts of the pattern. Cells in the last layer constitute the recognition layer with every cell

recognizing a particular pattern.

Every cell receives its inputs through a receptive field. For example, an S-cell does not search for a feature in the entire input plane. Instead, it is assigned a small area of the input plane. This area is called a receptive field. An S-cell fires only if the corresponding feature is available in its receptive field. The same concept is applied to C-cells where every C-cell has a corresponding receptive field that is different from other cells' receptive fields. Receptive fields usually overlap. Sometimes, the overlap is large that several neighboring cells respond to the same feature with nearly the same response.

3.1.2 Feature Extraction and Error Toleration

Toleration of errors is not done once at some layer. Instead, errors are tolerated (in a sense corrected) at every layer. A C-cell receives inputs from a group of S-cells. As discussed before, every cell from this group extracts the same feature, but from a slightly different position. If any S-cell from this group fires, the leading C-cell fires. Therefore, even if the stimulus pattern is shifted, the C-cell will still fire, provided that at least one of the corresponding S-cells fires [FW91].

Figure 3.3 shows an example of a neocognitron that finished training. Character 'A' is presented to the network. Every plane in the U_{S1} layer is responsible for

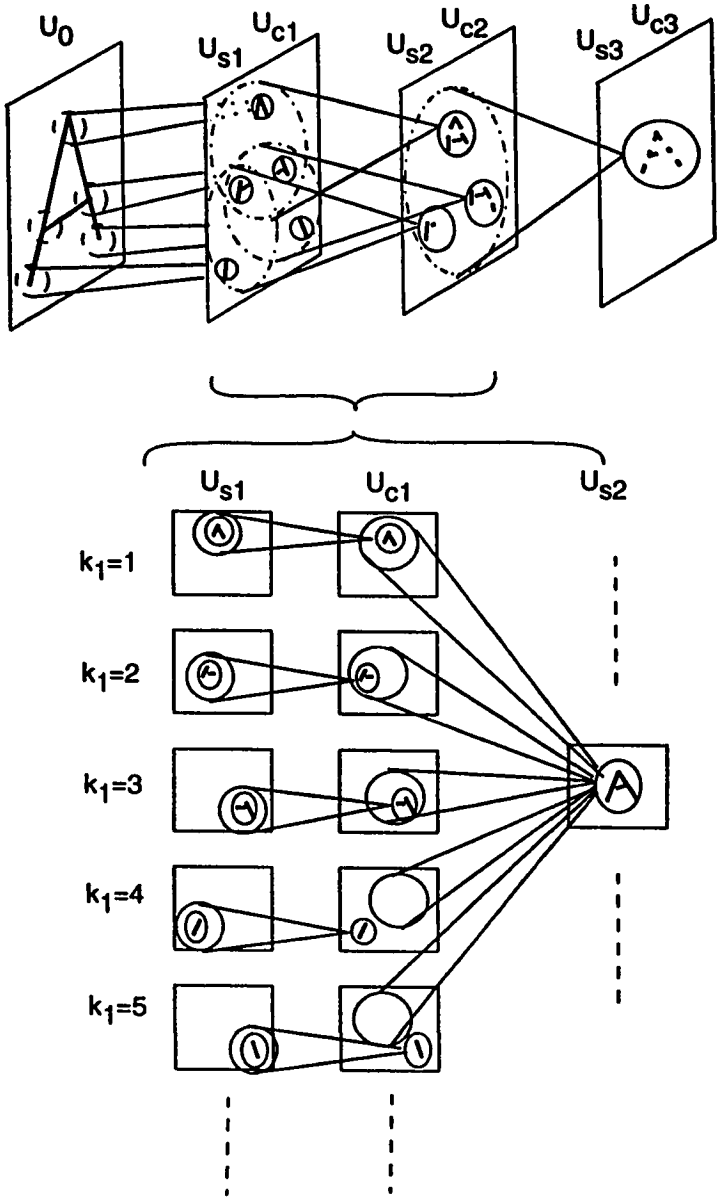


Figure 3.3: An example of the response of the cells after the completion of the training. From [Fuk89]

detecting a particular feature from the character 'A'. C-cells in the U_{C1} layer correct for shifts in the input character. However, some features ($k_1 = 4$ and $k_1 = 5$) can not be detected by the C-cells, because the amount of shifts that they undergo is beyond the receptive fields of the C-cells, as shown in Figure 3.3.

3.2 Interconnections of The Network

3.2.1 Connections of S-cells

S-cells are feature extracting cells. The concept of features is the same as that of primitives in structural pattern recognition. A feature constitutes part of the input pattern. This feature can be as small as a short stroke and it can be as large as the whole pattern. After the learning finishes, an S-cell is activated whenever a particular feature in the input pattern is present. These features are determined during the training process [Fuk88b]. Determining which S-cell should be activated when a particular feature is present can be done in two ways. In a supervised mode, the teacher decides on the S-cell that must fire for a particular feature [FW91]. In the unsupervised mode the network self-organizes. During training, the cell that has the highest output for a particular feature, will have its incoming connections reinforced. When a different feature is present, another S-cell will have the highest

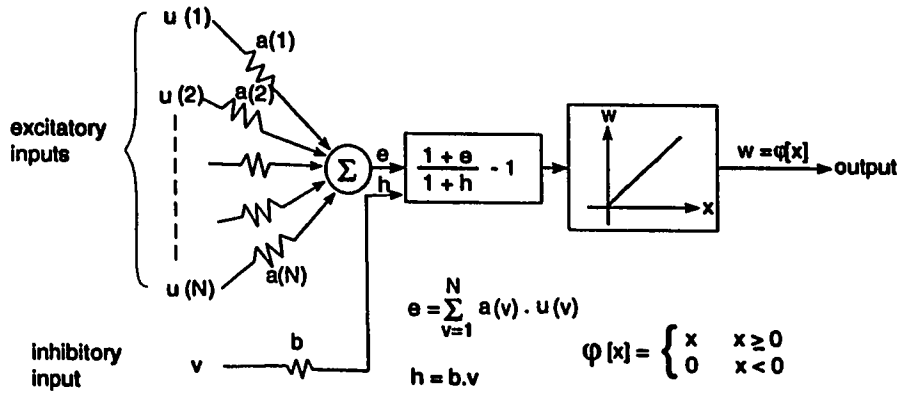


Figure 3.4: Input to output characteristics of S-cell. From [FM82]

output and will have its interconnections reinforced [Fuk88b]. In general the weights of the connections leading to an S-cell never decrease. Reinforcement of a weight involves increasing its value by an amount proportional to the response of the leading C-cell, and a preset constant.

An S-cell receives input from a group of C-cells in the preceding layer. These connections are variable and excitatory; they stimulate the output of the S-cell. In addition, an S-cell receives an inhibitory input from a V-cell, as shown in Figure 3.4. Suppose that the S-cell receives excitatory input from a group of C-cells whose output are : $u_1, u_2, ..., u_n$, and an inhibitory input from a V-cell whose output is v, then the output of the S-cell as expressed in [Fuk89] is :

$$u = r\varphi\left[\frac{1+\sum \alpha_i u_i}{1+\frac{r}{1+r}bv} - 1\right] \tag{3.1}$$

where v is the output of the V-Cell, r is a positive constant determining the efficiency of the inhibition by the V-Cell, α_i's are the weights of the interconnections, and b is the weight of the inhibitory connection that is modified during training phase. $\varphi[x]$ is defined as :

$$\varphi[x] = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.2}$$

.

3.2.2 Connections of C-Cells

A C-cell receives input from a group of S-cells in the preceding layer. These connections are fixed and determined beforehand. The values of these connections are determined so as to decrease with respect to the size of the connecting area [FMi83]. C-cells serve the function of error toleration. As described above, a C-cell gets its inputs from a group of S-cells in the preceding layer. When any one of these S-cells fire, this C-cell fires accordingly. If the position of the feature is shifted a bit, another S-cell will fire. Since the C-cell is connected to a group of S-cells, there is a good chance that this same C-cell will fire. This means that the neocognitron is shift invariant. Since scaling a pattern involves shifting its main features away

from each other, we would expect (as it is the case) that the neocognitron is scale invariant. In summary, the response of the C-cell results in a lower sensitivity to shifts in position and to scaling of the input feature [FW91].

3.2.3 Connections of V-Cells

Although it is not shown in Figure 3.3, every layer has a plane of V-cells. The dimensions of this plane are equal to the dimensions of the planes of the S-cells. In effect every S-cell receives an inhibitory input from its corresponding V-cell.

A V-cell receives its inputs from the same C-cells group as does the S-cell. These connections have fixed and predetermined values. These values are determined so as to decrease monotonically with respect to the size of the connecting group. The output of the V-cell is equal to the weighted average intensity of the response of the C-cells [FM82]. It is defined as :

$$v = \sqrt{\sum_i c_i {x_i}^2} \tag{3.3}$$

V-Cells play an important role in the recognition process. After the training finishes, every S-cell is trained to recognize a particular aspect of the pattern. If an S-cell is not supposed to respond to a particular feature, the V-cell is going to produce an output that will inhibit the S-cell from responding. In this way, V-cells

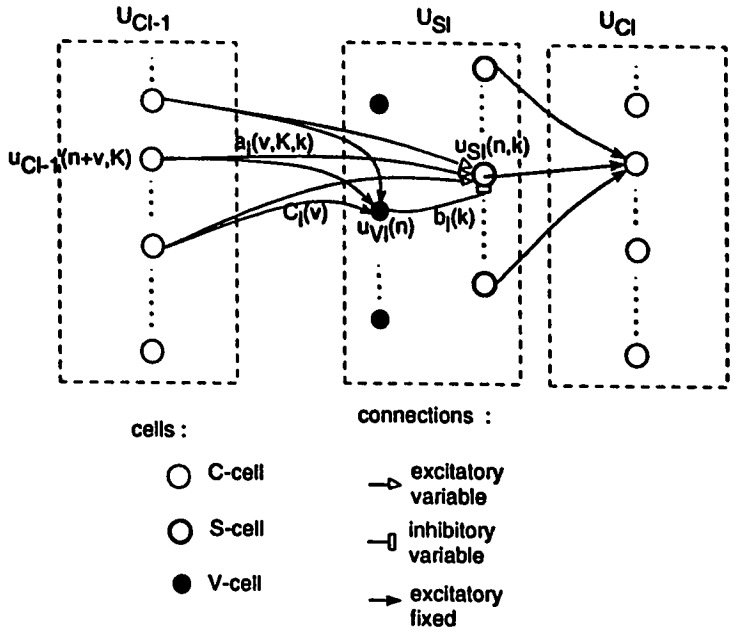


Figure 3.5: Connections between cells in the network. From [Fuk89]

help the S-cells in differentiating irrelevant patterns [Fuk89]. Figure 3.5 shows an example of the connections between the three different cells.

3.3 Training of the Network

Training of the network is done in stages. Training of a particular layer does not commence until the training of the preceding layer finishes [FW91]. During the training of a particular layer, the cells in this layer will produce outputs with different intensities. If the outputs of these cells are allowed to affect the response of the succeeding layer's cells, the connections to the succeeding layer's cells might be

erroneously reinforced. In order to prevent such a situation, the outputs of the layer being trained are retarded and prevented from affecting the succeeding layer [FM82].

3.3.1 Supervised vs. Unsupervised Training

As discussed above, the network can be trained under supervision, or it can be allowed to self-organize. Under unsupervised training, a pattern is presented to the network. The cell that produces the maximum output is selected as the seed cell. In one plane there can be no more than one seed cell. The seed cell will have its incoming connections reinforced. When another feature is presented, another cell from a different plane will have the highest output. This cell will have its interconnections reinforced so that it can detect this feature later [FW92].

Upon this reinforcement of the connections of the seed cell, it will fire whenever its corresponding feature is present in its receptive field. In addition, all the cells in the same plane will have their incoming connections reinforced in the same manner. In this way, every plane is going to detect a particular feature from the input pattern. Even if the pattern is shifted a bit, other cells in the same plane will fire, and in effect, detect that feature. [FW92]

Under supervised training, the teacher selects a cell plane to be trained. He then presents a training pattern to the network and decides on a cell from this plane to

be appointed as a seed cell. Usually, the center cell is selected as a representative. The incoming connections to this seed cell are reinforced. The amount of the reinforcement depends on the intensities and the weights of the incoming connections. As in unsupervised training, all other cells in the same plane will have their interconnections reinforced in the same manner so as to respond to the same feature. [FW91]

A decision concerning the training method is influenced by the number and the complexity of the characters to be recognized. For the case of supervised training, constructing a good training set requires a lot of time especially when the number of characters to be recognized is large or when the complexity of the characters is high [FW91]. The reason is that the teacher has to determine the positions at which important features exist. In addition, determining important features that distinguish between characters is a tricky task that depends to a large extent on the creativity of the teacher [FW92]. Every layer in the network has a training set that differs from other layers' training sets. Training sets not only include the complete characters, but also small local features which are important for the training of intermediate layers. This results in a robust network. However, when unsupervised training is employed, where the network is left to self-organize without guidance, the trained network is not robust enough to detect some deformed patterns [FW92].

Supervised training has another advantage: once the training set is determined,

training of the network can be done in a very short time. A comparison is presented in [FW91] in which training of the neocognitron was shown to take 13 minutes on the SUN station. Another network using backpropagation training consumed three days of training time on a SUN workstation.

The need for supervised training arises when patterns of different shapes need to be mapped to the same category. An example is when the same character might be present in the document but with different fonts [FW92] or with a different shape as a result of a different writing style.

3.3.2 Update of Synapses Weights

As discussed above, we have two types of *variable* connections, namely: connections leading from C-cells in a layer to an S-cell in the succeeding layer, and connections leading from a V-cell (the inhibitory cell) to its corresponding S-cell.

The amount of reinforcement, Δa_i, of the connection that leads to an S-cell is defined as:

$$\Delta a_i = q c_i x_i \tag{3.4}$$

where c_i is the weight of the connection leading from the ith C cell to the V-cell. x_i is the intensity of the output of the ith C-cell, and q is a positive constant determining

the speed of reinforcement. In the case of unsupervised training, q usually takes small values (< 2.0). This is because of the large number of repetitions that the training undergoes. However, in the supervised training a cell plane is trained only once. As a result, q is assigned a large value (> 1000).

The amount of reinforcement, Δb, of the connection leading from a V-cell to an S-cell is defined as:

$$\Delta b = q\sqrt{\sum_i c_i \{x_i\}^2} \tag{3.5}$$

3.4 Network Setup

The complexity of the characters to be recognized influences the number of layers of the network. If the characters are complex, features of small sizes have to be detected. This requires having small receptive fields of S-cells. As a consequence, the number of stages (or layers) of the network has to increase [FW91].

The number of planes required in every layer is made equal or larger than the number of features to be detected at that particular layer.[FW91]

Two parameters control the training and the performance of the network, namely: r and q. Parameter r, which is called the selectivity controller, determines the efficiency of the inhibition by the V-cells. In a sense, it determines the selectivity of

the network as can be seen from the following equation:

$$u = r\varphi\left[\frac{1+\sum a_i x_i}{1+\frac{r}{1+r}bv} - 1\right] \tag{3.6}$$

A large value of r favors the inhibition of the V-cell, and in effect corresponds to a smaller tolerance of noise and deformation of the feature. One must not be misled by thinking that a small value of r will endow the network with a good toleration of noise. A very small value of r will make the network erroneously detect irrelevant features [FMi83].

The parameter q is very important in *self-organization* of the network. As discussed above, while training a particular layer it is important to retard the outputs of this layer from affecting the succeeding layer. A careful selection of the value of q allows for correct training of the network. A network with small number of layers, one or two layers for example, can have a large value of q. As the number of layers increases, the value of q must be small enough to retard the outputs of any layer being trained from affecting the succeeding layers [FM82]. In the supervised mode, this parameter can have large values, because every layer needs one cycle to train.

3.4.1 Selectivity of S-cells

As mentioned above, the selectivity of an S-cell is controlled by the parameter r. In the following section, we will clarify the effect of this parameter.

Let $P(v)$ be the pattern used to train a particular plane, and let $p(v)$ be the pattern that will stimulate the response of the representative cell of this plane. Let cell u be chosen as a representative of this S plane. The excitatory weights of this cell will be updated by the following amount :

$$\Delta a(v) = q \cdot c(v) \cdot P(v) \tag{3.7}$$

and the inhibitory weight will be updated by the amount:

$$\Delta b = q \cdot v \tag{3.8}$$

When running the network, an S-cell responding to the pattern,$p(v)$, will have the output:

$$u = r \cdot \varphi \left[\frac{1 + \sum_v a(v) \cdot p(v)}{1 + \frac{r}{1+r} \cdot b \cdot v} - 1 \right] \tag{3.9}$$

and the inhibitory output is defined as :

$$v = \sqrt{\sum_v c(v) \cdot p^2(v)} \tag{3.10}$$

For simplicity of discussion and provided that $a(v)$ and b are sufficiently large (by choosing a large value for q, these weights will have large values), equation (3.9) simplifies to :

$$u = r \cdot \varphi \left[\frac{r+1}{r} \cdot s - 1 \right] \tag{3.11}$$

where s is defined as :

$$s = \frac{\sum_v a(v) \cdot p(v)}{b \cdot v} \tag{3.12}$$

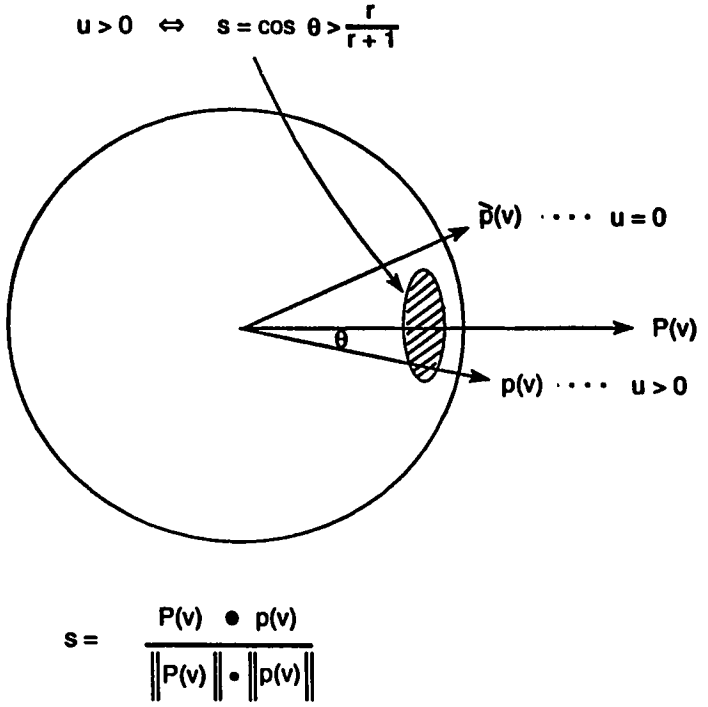


Figure 3.6: Similarity between patterns. From [Fuk89]

and

$$\varphi[x] = max(x, 0) \tag{3.13}$$

The variable s represents a measure of the similarity between the training pattern $P(v)$ and the stimulus pattern $p(v)$. When s is larger than $\frac{r}{1+r}$, the S-cell starts responding. Substituting the values of $a(v)$, v, and b in (3.12) we obtain:

$$s = \frac{\sum_v c(v) \cdot P(v) \cdot p(v)}{\sqrt{\sum_v c(v) \cdot P^2(v)} \cdot \sqrt{\sum_v c(v) \cdot p^2(v)}} \tag{3.14}$$

Equation (3.14) defines s to be the weighted inner product of two vectors nor-

malized by their weighted norms [1]. This inner product equates to the *cosine* of the angle between the two vectors. Therefore, s can take values within the range $0 \leq s \leq 1$. When the vectors $p(v)$ and $P(v)$ are equal, s becomes maximum and have a value 1. When $p(v) \neq P(v)$,we have $s < 1$. As can be seen from equation (3.11), when s falls within the range : $0 \leq s \leq \frac{r}{1+r}$, the S-cell outputs 0. It starts responding when $s > \frac{r}{1+r}$, and have a maximum value of 1 when $s = 1$.

Figure 3.6 interprets the above discussion geometrically. The stimulus pattern elicits a response from the S-cell u if it falls within the shaded area. The size of the shaded area can be controlled by r. By increasing the value of r the size of the shaded area gets smaller, which means that the S-cell will be less tolerant to deformations or noise.

3.4.2 Selection of Selectivity Controllers

Fukushima in his paper [FM82] derived a formula for the calculation of the selectivity controllers, r. His derivation starts with equation (3.9). Assuming that $c(v)$'s are the same for all values of v, and that $P(v)$ and $p(v)$ have binary values, we have :

$$s = \frac{1 - \frac{n_0}{N}}{\sqrt{1 + \frac{n_1 - n_0}{N}}} \tag{3.15}$$

[1] The norm of a vector is its length

where n_0 is the number of elements which are active in $P(v)$ but have zero values in $p(v)$. n_1 is the number of elements which are active in $p(v)$ but have zero values in $P(v)$. N is the number of active elements in the training pattern, $P(v)$. As was shown before, when r satisfies the condition : $s > \frac{r}{1+r}$, the corresponding S-cell starts responding. Substituting the value of s in this condition we have :

$$\frac{1 - \frac{n_0}{N}}{\sqrt{1 + \frac{n_1 - n_0}{N}}} > \frac{r}{1+r} \tag{3.16}$$

Selection of r for the first layer

Every plane in the first S-layer of the network is trained on a 3 × 3 pattern. Every pattern of those have exactly three active elements. We want every plane in this layer to tolerate not more than one additive element and one subtractive element. The plane must not tolerate the absence of two elements or more.

Toleration of one additive and one subtractive element translates to the following assignments: $n0 = 1$, and $n1 = 1$. Substituting these values in equation (3.16) we find that r falls in the range, $r < 2$. Toleration of two additive and one subtractive translates to $n_0 = 1$ and $n_1 = 2$. Substituting these values in the equation we get the following range for r: $r < 1.4$. If we are to allow the plane to tolerate two subtractive (which we really do not want) and no additive elements the range of r will also be: $r < 1.4$. Consolidating these results, we get the following permissible

range for r: $1.4 \leq r < 2.0$. We choose to assign the intensity controllers of the first layer a value of 1.60. This assignment will be justified in Section 4.5.1.

Selection of r for the Remaining Layers

Equation (3.16) can only be applied to binary patterns. In layers higher than the first one, a particular plane is normally trained on several training patterns rather than one. It is not possible to map training on several patterns to training on only one binary pattern. In addition, layers higher than the first one have their inputs processed by the first layer. The first C-layer generates real values in the range: 0 .. 1. Since the second S-layer gets its inputs from the first C-layer, the notion of binary training patterns can not be applied in this context. It is left for experimentation to decide on the optimum values for r's.

3.4.3 Planes Dimensions

Selection of a particular plane size can affect to a large degree the performance of the system. Consider the example shown in Figure 3.7 which shows part of the network. Let's assume for simplicity, that every layer has one plane only. The Figure shows one dimension of every plane. The Figure also shows a feature which is the top part of the hindi character '4' as a training pattern. The training pattern is used to train

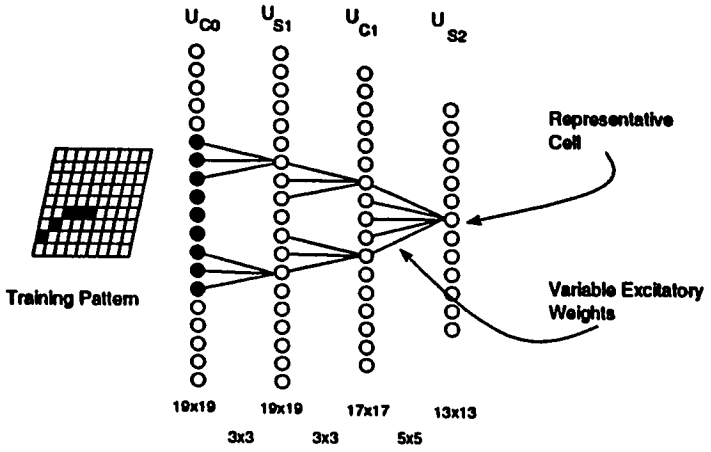


Figure 3.7: The network's partial configuration during the training stage

a particular plane in U_{S2} whose representative is labeled in the Figure.

In this setup, cell 1 in U_{S2} will have its receptive field covering the first *nine* cells in U_{C0}. The Figure also shows the dimensions of the planes and the sizes of the receptive fields. As explained above, when training of U_{S2} finishes, all cells in the plane will share the same 5×5 excitatory weights, shown in the Figure.

Figure 3.8 shows an example of the network stimulated by character '4'. Unfortunately, when the character '4' is presented to the network, its top feature will not be detected. The network measures the similarity between a feature in the stimulus pattern and the feature on which it was trained by overlapping the two features and finding how well they correlate. The solid frame represents the target field which must be achieved in order for the enclosed feature to correlate well with the training feature. The dashed frame represents the maximum achieved field using the setup

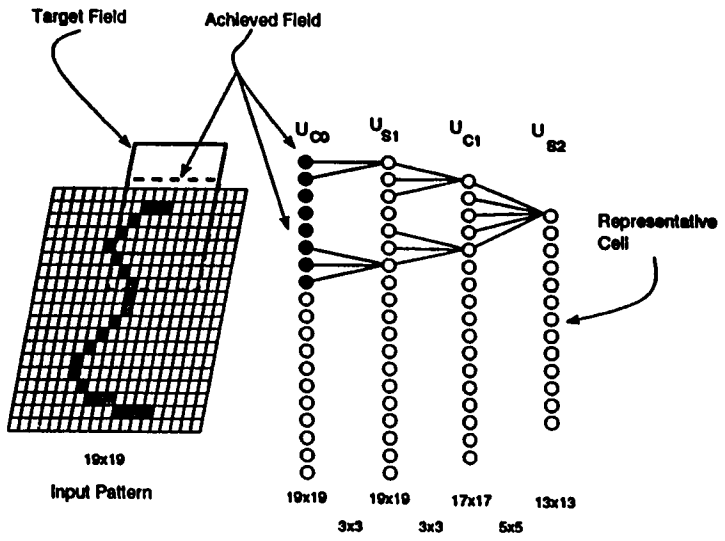


Figure 3.8: The network during operation

plotted in Figure 3.8.

As can be seen in the Figure, the first cell in U_{S2} covers cells : -1 through 7 in the input plane. Of course, there is no cell with index -1 in the input plane. This means that the receptive field extends one pixel outside the input plane. We want this cell to cover cells : -4 .. 4. We need a cell whose receptive field not only covers part of the pattern (or the input plane), but also extends *four* pixels (maximum) to the outside. Using the current setup of the network, there is no cell with this property.

A solution to this problem would be endowed by increasing the sizes of the planes especially the planes of S-layer 2. Figure 3.9 shows the network after modification.

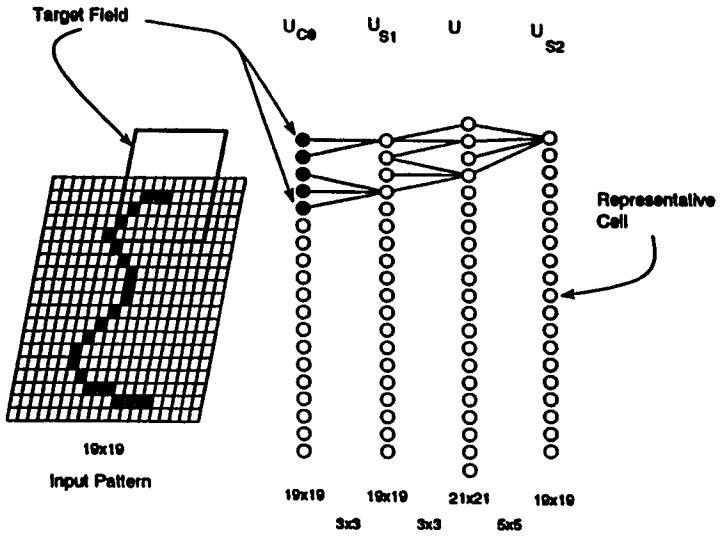


Figure 3.9: Modification of the network in order to solve the problem

planes in U_{C1} and U_{S2} now have larger dimensions. Using this new setup, the first plane in U_{S2} can have the target receptive field and the feature at the top of the character will be detected.

3.4.4 Determining Sizes of Receptive Fields

The sizes of S-fields and C-fields are influenced by two factors discussed in [FW91]:

- The density of local features in the input patterns. When the density of features is high, small receptive fields are needed to detect them, and to segregate different features. When the local features are sparse, the receptive fields can be large.

- The degree of deformation of the patterns. As in the previous factor, the degree of deformation is inversely related to the fields sizes. If the deformation is small, the receptive field can be large. If large deformation has to be tolerated, the sizes of the receptive fields must be small.

Fukushima made good use of the these two factors in designing all of his networks. In Figure 3.10 we show an example of a network designed by Fukushima for the recognition of arabic numerals. The Figure shows a one dimensional view of the interconnections between cells of different planes. Only one cell-plane is drawn in each layer. Using the same architecture specifications and exact parameters used by Fukushima, we implemented the design that he used for the recognition of the arabic numerals. However, the performance of this network was very weak. The recognition rate was 37% while the confusion rate was 55%. Two reasons affected the performance :

- System parameters change from one set of patterns to another
- The receptive fields were not designed properly and they did not have the correct sizes

When deciding on the sizes of the receptive fields, one has to consider the sizes of the training patterns. Every layer in the network has a set of training patterns with

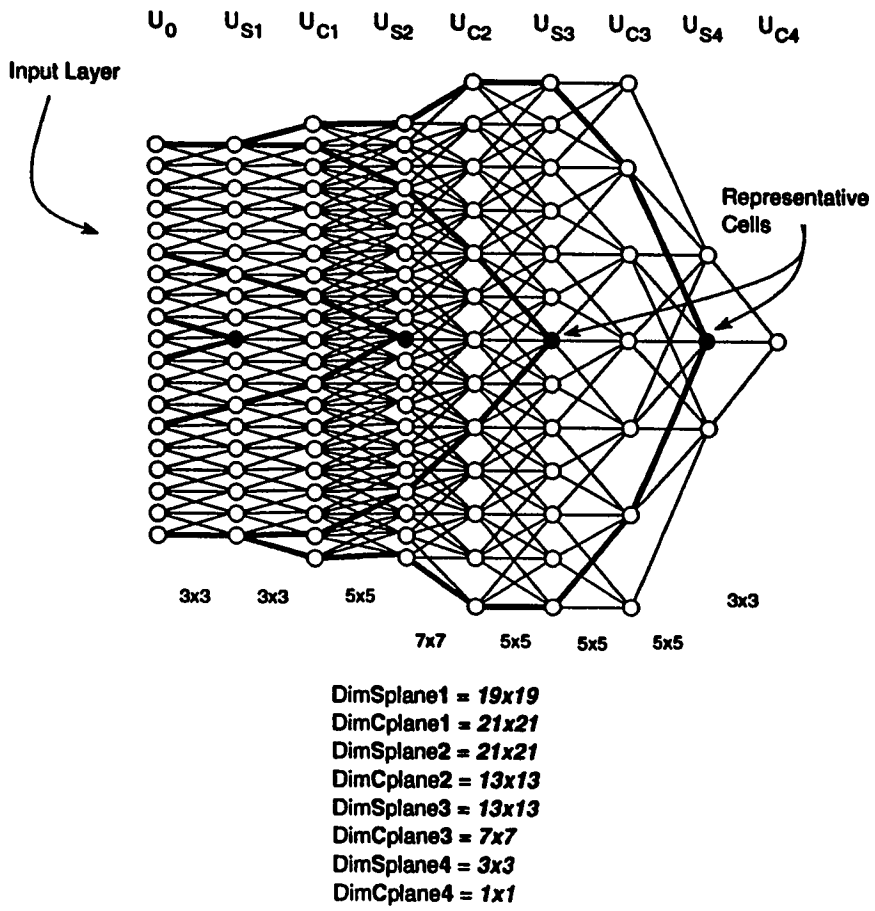


Figure 3.10: Network setup designed by Fukushima

a particular size that is different from other layers'. The dimensions of the training patterns for different layers are shown in the following table :

Layer	Size of Training Patterns
1	3x3
2	9x9
3	19x19
4	19x19

As mentioned above, during the process of training the network, a representative cell is selected from every plane. Usually the cell at the center is selected as a representative. The training pattern is centered in the input plane. The response of all cells in the layers preceding the layer of the representative are computed. The variable excitatory connections of the plane undergoing training are updated accordingly.

U_{S1} is trained on 3×3 patterns forcing the S-fields of this layer to be of the same size. U_{S2} is trained on 9×9 patterns. Having S-fields of 5×5 size would allow the representative cell to cover these training patterns, as shown by the solid lines in Figure 3.11.

This figure displays a very subtle problem in the design of Fukushima. The receptive field of U_{S2} is 5×5 pixels in size. The dimensions of the field were selected

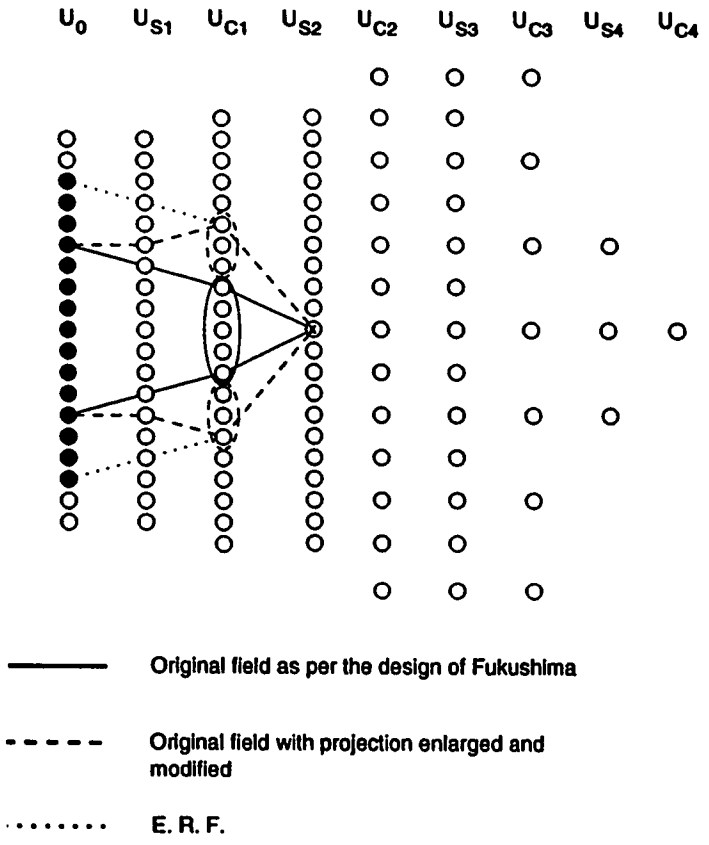


Figure 3.11: Comparison between different field settings

so that they will cover the entire training pattern. As shown in the Figure, the field does cover the training pattern. However, this receptive field does not cover all of the responses generated by the training pattern. The S-cells in U_{S1} can act as edge detectors, because as mentioned above they can tolerate three additive elements. Therefore, when a 9×9 pattern is presented, it is not that only seven cells in U_{S1} will respond. In fact, nine cells (which are confined by the two dashed lines) will produce outputs. Continuing in this manner, we can see that it is not that only five cells in U_{C1} (confined in the solid oval) will produce outputs. In this layer, six more cells will respond (confined in the two dashed ovals,) bringing the number of responding cells to 11.

An intuitive solution to this problem will be to extend the receptive field of U_{S2} to become 11×11. This easy solution does not come without any compromise. Extending the receptive field will make the representative cell cover larger area in the input plane than what it should actually cover. As a result, the outputs of 15 cells in the input plane will be considered in calculating the response of the representative cell of U_{S2}.

If the receptive field size of U_{S2} retains its original size, (5×5), the representative cell would not cover all of the necessary cells in the previous C-layer. If the size of the receptive field is extended, the representative cell will cover unwanted cells in the input plane. It is clear from the above discussion that these two problems are

inherent in the architecture of the neocognitron. Experimenting with the two setups, we found that extending the receptive fields with careful parameter selection allows the network to perform much better than retaining their original sizes. Besides the issue of field sizes is crucial mostly to the second layer.

Chapter 4

Design and Analysis Issues

This chapter presents the basic design issues keeping in mind the theoretical issues mentioned in the previous chapter. The setup of the network is presented along with the complete parameters assignments. Extensive analyses are carried out for every parameter. This is to ensure that the parameter is assigned the correct value and to capture the general trend of the performance versus different values for this parameter. In addition, the training set is presented along with the necessary explanation.

4.1 Network Operational Description

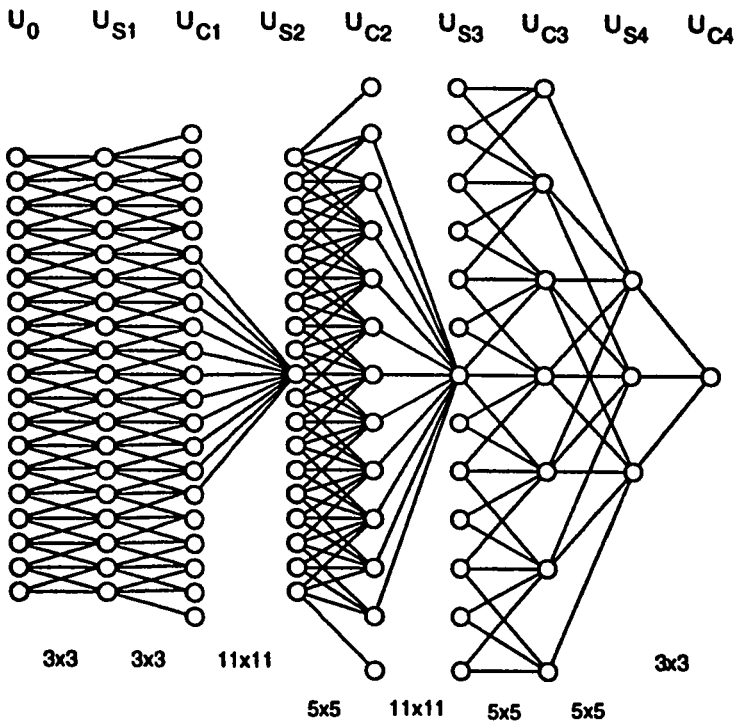
In this section, a brief mathematical description of the running of the network is presented; where all the necessary mathematical equations are consolidated. All of these equations have been presented in previous chapters, but they are repeated here in full detail. The section is meant to be self contained; enabling the reader to build and run his desired network in a short time.

The job of the network's designer consists of four parts:

- Designing the training set
- Selecting the correct parameters
- Training the network
- Running the network

The training set is presented Section 4.3. Figure 4.1 depicts the general network architecture, along with the entire parameters setup. The remaining part of this chapter is dedicated mostly to the analysis of these parameters.

During the training phase, training patterns are presented to the system. The teacher presents a training pattern and decides on a plane for this pattern. He



DimSplane1 = *19x19*
DimCplane1 = *21x21*
DimSplane2 = *19x19*
DimCplane2 = *13x13*
DimSplane3 = *13x13*
DimCplane3 = *7x7*
DimSplane4 = *3x3*
DimCplane4 = *1x1*

FIELDS1 = *3*
FIELDC1 = *3*
FIELDS2 = *11*
FIELDC2 = *5*
FIELDS3 = *11*
FIELDC3 = *5*
FIELDS4 = *5*
FIELDC4 = *3*

Q1 = *1000*
Q2 = *1000*
Q3 = *1000*
Q4 = *1000*

BASE = *0.9*
ALPHA = *0.7*

Figure 4.1: System setup and parameters assignments

then selects a particular cell from this plane as a representative. Let cell $u_{Sl}(\hat{n}, j)$ be selected as a representative, the excitatory weights of this S-plane are updated according to the equation:

$$\Delta a_l(v, i, j) = q_l \cdot c_l(v) \cdot u_{Cl-1}(\hat{n} + v, i) \tag{4.1}$$

where $\hat{n}$ is the coordinate of the S-cell of plane j of layer l that has been selected as a representative for this plane. The variable v denotes a two dimensional receptive field, i denotes a particular C-plane preceding this S-layer that is being covered by v, and j denotes the S-plane whose excitatory weights are being reinforced. The speed of reinforcement is denoted by q_l. $c_l(v)$ signifies the set of fixed excitatory weights connecting to the V-cell and $u_{Cl-1}(\hat{n} + v, i)$ is the output of U_{Cl-1}-cell$_{\hat{n}+v}$.

In addition, the inhibitory variable connection leading from a V-cell to its corresponding S-cell [1] is updated according to the following:

$$\Delta b_l(j) = q_l \cdot u_{Vl}(\hat{n}) \tag{4.2}$$

The fixed excitatory connection $c_l(v)$ is a monotonically decreasing function of $|v|$ and it is initialized according to the equation:

$$c_l(v) = BASE^{|v|} \tag{4.3}$$

[1] Every S-cell has one corresponding V-cell

where $BASE$ takes a value of 0.9 in our implementation and $|v|$ is defined as :

$$|v| = \sqrt{x^2 + y^2} \tag{4.4}$$

where x and y represent the dimensions of v. Once the training of layer U_{Sl} finishes, the output of every S-cell in this layer is computed according to the formula :

$$u_{Sl}(n,j) = r_l \cdot \varphi \left[\frac{1 + \sum_{i=1}^{K_{Cl-1}} \sum_{v \in A_l} a_l(v,i,j) \cdot u_{Cl-1}(n+v,i)}{1 + \frac{r_l}{1+r_l} \cdot b_l(j) \cdot u_{Vl}(n)} - 1 \right] \tag{4.5}$$

where

$$\varphi[x] = max(x, 0) \tag{4.6}$$

and A_l denotes the summation range of v. As discussed above, all the S-cells in a plane share the same set of excitatory weights. Therefore, $a_l(v,i,j)$ and $b_l(j)$ do not contain the argument n denoting the location of the receptive field with respect to the S-plane.

The subsidiary V-cell which sends an inhibitory signal to its corresponding S-cell produces an output according to the formula:

$$u_{Vl}(n) = \sqrt{\sum_{i=1}^{K_{Cl-1}} \sum_{v \in A_l} c_l(v) \cdot \{U_{Cl-1}(n+v,i)\}^2} \tag{4.7}$$

The output of U_{Cl}-cell$_n$ is computed as per the following equation:

$$u_{Cl}(n,i) = \psi \left[\sum_{j=1}^{K_{Sl}} conn_l(j,i) \sum_{v \in D_l} d_l(v) \cdot u_{Sl}(n+v,k) \right] \tag{4.8}$$

where $\psi[x]$ is a saturation factor defined as :

$$\psi[x] = \frac{\varphi[x]}{ALPHA + \varphi[x]} \tag{4.9}$$

Some S-planes have their outputs joined into one C-plane. The variable $conn_l(j,i)$ specifies whether there is a connection between S-plane$_j$ and C-plane$_i$ or not. The fixed excitatory connections representing the receptive field of a C-cell are represented by $d_l(v)$.

4.2 Preprocessing Stage

4.2.1 Thresholding the Character Images

Images of characters are taken using a normal video camera. An image is saved as a bitmap file (*.BMP.) Every pixel in the image is stored as a gray-level value in the range : 0 .. 255. Our objective is to transform the gray-level image into a binary image. A method for gray-level thresholding in badly illuminated images is presented in [Par91]. However, this method works well only when an added illumination, whether linear gradient or gaussian, is present. Besides, for every image, this method requires careful selection of some parameters in order to obtain optimum results. The threshold selection method that we used is based on the gray level histogram of the image, presented in [Ots79]. In this section, we present the details of this algorithm along with its application to our system.

Let the number of gray levels in the image be L. The number of pixels having a

gray level i is denoted by n_i. The total number of pixels in the image is defined as $N = n_1 + n_2 + ... + n_L$. The probability density of the gray levels is defined as :

$$p_i = n_i/N, \qquad p_i \geq 0, \sum_{i=1}^{L} p_i = 1 \tag{4.10}$$

Suppose that we select a particular gray level value, k, as a threshold, then all pixels with gray levels less than or equal to k are considered as 0's and the rest of pixels are considered as 1's. We define the class probabilities as :

$$\omega_0 = \sum_{i=1}^{k} p_i = \omega(k) \tag{4.11}$$

$$\omega_1 = \sum_{i=k+1}^{L} p_i = 1 - \omega(k) \tag{4.12}$$

The class mean levels are defined as:

$$\mu_0 = \sum_{i=1}^{k} i \cdot p_i/\omega_0 \tag{4.13}$$

$$\mu_1 = \sum_{i=k+1}^{L} i \cdot p_i/\omega_1 = \frac{\mu_T - \mu(k)}{1 - \omega(k)} \tag{4.14}$$

where

$$\omega(k) = \sum_{i=1}^{k} p_i \tag{4.15}$$

,

$$\mu(k) = \sum_{i=1}^{k} i \cdot p_i \tag{4.16}$$

and the total mean level of the image is

$$\mu_T = \sum_{i=1}^{L} i \cdot p_i \tag{4.17}$$

In order to measure the *goodness* of the selected threshold, OTSU introduced a measure called *between class variance*. This measure is defined by

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

$$= \omega_0\omega_1(\mu_1 - \mu_0)^2 \tag{4.18}$$

The problem of thresholding reduces to finding a value for k which maximizes σ_B^2. By substituting equations (4.11),(4.12),(4.13),(4.14) into (4.18) we get:

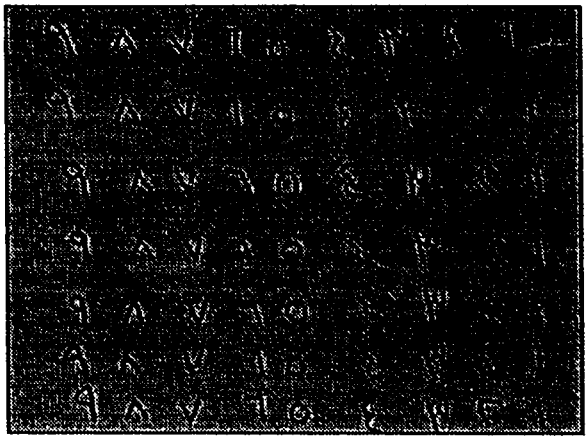
$$\sigma_B^2(k) = \frac{[\mu_T\omega(k) - \mu(k)]}{\omega(k)[1 - \omega(k)]} \tag{4.19}$$

and the optimal threshold k^* is chosen such that

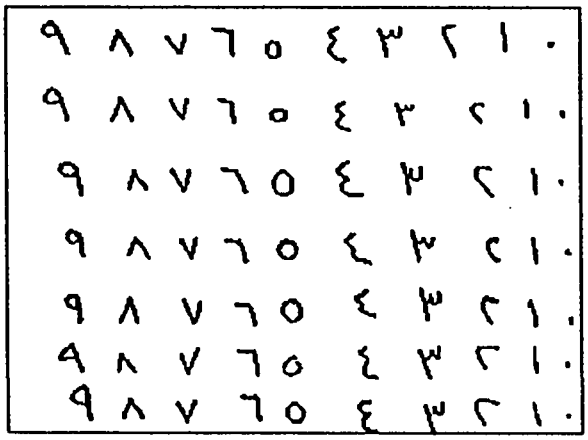
$$\sigma_B^2(k^*) = \max_{1\leq k<L} \sigma_B^2(k) \tag{4.20}$$

Figure 4.2 shows an example of a gray-level image and its result after thresholding it. In original image, the average gray level is 128.5. However, as can be seen from Figure 4.3, the *between class variance* is maximized at the gray-level value 101. This value was selected as the threshold.

We can see from the histogram plot that we only have one peak which is corresponding to the white pixels. However, there is no peak for the black pixels. This can be explained by the nature of the image. Our images are of hand-written characters. The width of the character stroke does not usually exceed 10 pixels. Normally, the



a) Original Image



b) Binary Image

Figure 4.2: Gray level image and the corresponding binary image

gray-levels of the pixels of the character stroke change at the boundaries. The gray-levels stabilize around the center of the line crossing the width of the stroke. Since the strokes are small, we do not expect to find an area of stability of the gray-levels. There will be no particular gray-level that has a very large frequency compared to the rest of the gray-levels. As a result, the distribution of the frequencies of the black pixels is close to constant, which explains the absence of a peak corresponding to the black pixels.

4.2.2 Line Thinning

In order to test patterns, Fukushima used a magnetic tablet on which characters were drawn. His system was designed to handle characters with one pixel width. In order to accomplish this task, the tablet was divided into 19 × 19 squares where each square represents a pixel. The size of the square is larger than the locus of the mouse pointer. A square is activated if the mouse pointer crosses the square more than half of the length of its side. In this way, the character's stroke is guaranteed to be one-pixel wide [FW91].

One problem with entering the characters via a camera is that the characters' images might differ in size considerably. This is because of two factors:

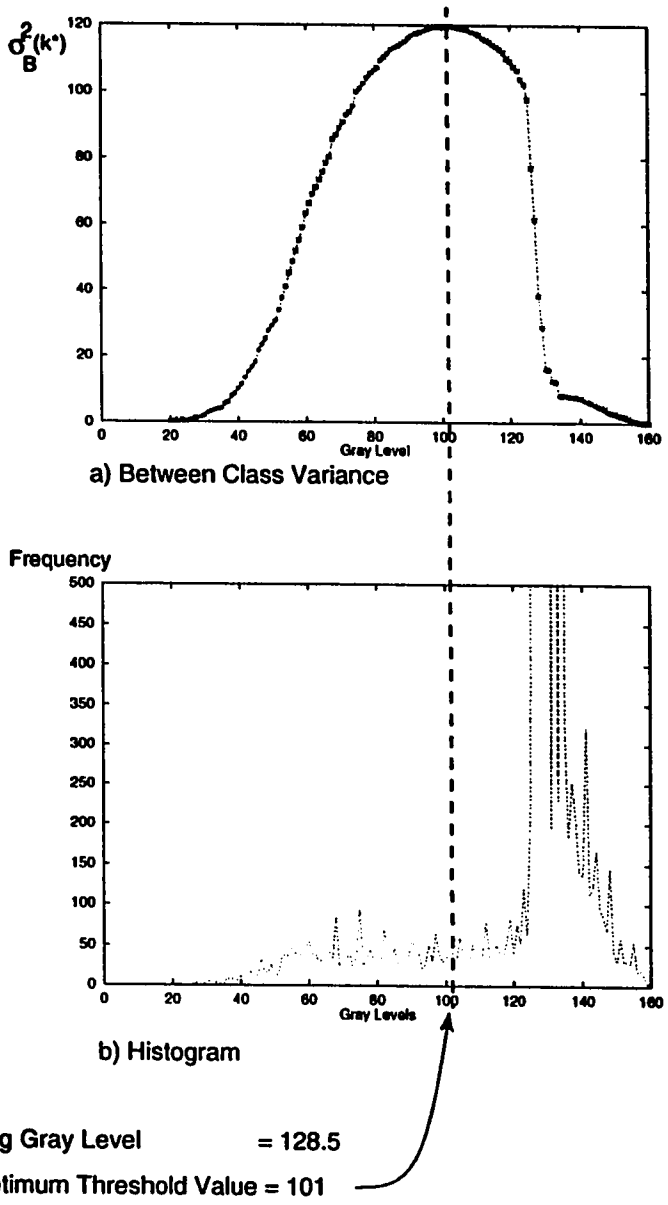


Figure 4.3: The between class variance and the histogram of the image

- The location of the camera: As the camera gets closer to the character, the character's image gets larger.
- There is no restriction on the pen that is used in writing the characters.

As a result of these two factors, characters will have varying stroke widths.

The size of the character stroke width has an impact on designing the training set. In the training set, we include important features that help in differentiating characters. In addition, we also include examples of the characters to be learned. During this stage, we have to think of an acceptable stroke width. Since mapping varying stroke width's to one-pixel width is the most common approach, we choose to follow this common convention.

If we are to allow the network to accept and recognize characters with different stroke widths, the training set has to be modified. Suppose that we allow the network to deal with characters whose stroke widths range from one to five. This implies that the training set has to be duplicated five times. Each subtraining set will have its training patterns drawn with a particular stroke width. Line thinning solves these problems by reducing the character to a line drawing character.

In addition to the benefit of standardizing the characters, line thinning can eliminate contour distortions while retaining the general topological structure of the

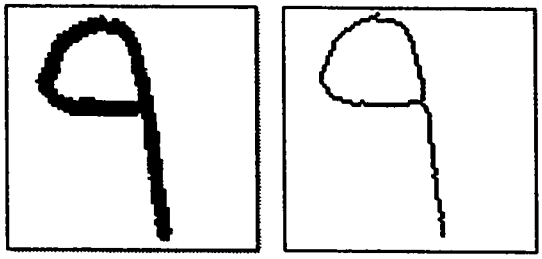


Figure 4.4: Example of a character before and after thinning is applied

character. Generally speaking, thinning a character to its skeleton eases the human conception of its structure. Besides, thinning a character eases the extraction of important features, such as end points, junction points, and connections between components.

SPTA Thinning Algorithm

The process of line thinning involves removing points along the contour of the character until the character is thinned to its skeleton [O'G90]. Figure 4.4 depicts the case of a character being thinned to a one-pixel line drawing.

The deleted points must not affect the connectedness or the shape of the pattern. Algorithms that remove points along the contour differ in the way they run tests on the points. However all thinning algorithms agree on three basic conditions, presented in [LLS92]:

1. the point is a dark pixel.

2. the point is not an end point

3. the point is a contour point.

In most of the algorithms, a 3 × 3 window is considered for testing. [O'G90] generalizes this concept and allows windows of size $k \times k$ where k is an arbitrary odd integer. Allowing k to take values larger than three would speed up the process of thinning. However, this speed up does not come with no compromise. A character thinned in this way would have a lot of spurs at the branches.

The algorithm that we used in the thinning process is proposed by [NS84]. The algorithm is called "safe point thinning algorithm" (SPTA.) This algorithm was chosen because it is the basis from which many algorithms have evolved. In addition, the algorithm is fast and produces good quality.

SPTA uses windows of 3 × 3 size. Points in this window are defined as follows:

n3	n2	n1
n4	p	n0
n5	n6	n7

where p is the point considered for deletion. An *8-neighbor* is any point around

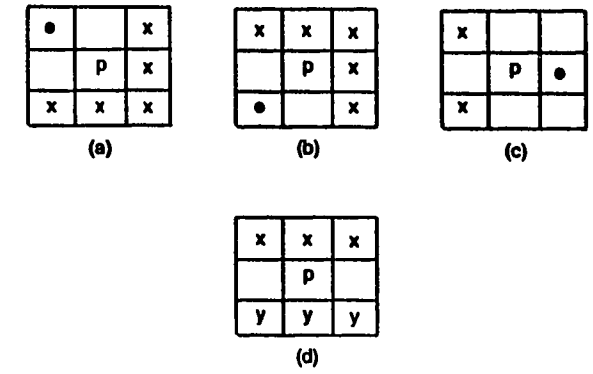


Figure 4.5: Safe points matching windows. From [NS84]

p. A *4-neighbor* is any of: $n0, n2, n4, n6$. A *left edge* point is any point the has $n4$ as a white point. A *right edge* point is any point that has $n0$ as a white point. The same applies to $n2$, and $n6$ in defining a *top edge point* and a *bottom edge point.*

An edge point is *safe* if it can not be deleted. A point can not be deleted if it falls in one of the following categories:

end point: a point that has at most one dark point among its eight neighbors.

break-point: a point whose deletion would break the connectedness of the pattern.

excessive erosion point: a point whose deletion causes excessive erosion of the pattern.

When the neighborhood of p matches any of the windows in Figure 4.5, p is

considered a safe point and it is not deleted. If the neighborhood of p matches any of windows (a), (b), or (c) of Figure 4.5, we have the following two situations:

- if all x's are whites, p is an end point
- if at least one x is dark, p is a break point,

and in both cases p must not be deleted.

If the neighborhood of p matches window (d), then we have the following. If at least one x and at least one y are black, then p is a break point and it can not be deleted. If all of the x's are white, we will have the following eight possible windows shown in Figure 4.6. Configurations w1,w2, and w3 make p an end point. configuration w4 makes p a break point. Configurations w5 and w6 could cause excessive erosions in slanting strokes of width 2. The existence of configurations w7 and w8 indicates the presence of a noise. Since the neocognitron performs well even with the presence of noise, we would not remove noise in any preprocessing stage. Therefore, in all of these case, p is considered a safe point and it is not deleted.

In order to check whether a left edge point, p, is a safe point or not, it is enough to check for the following boolean condition:

$$S_4 = n_0 \cdot (n_1 + n_2 + n_6 + n_7) \cdot (n_2 + \bar{n}_3) \cdot (n_6 + \bar{n}_5)$$

where a variable takes the value **TRUE** if its corresponding point is dark, and

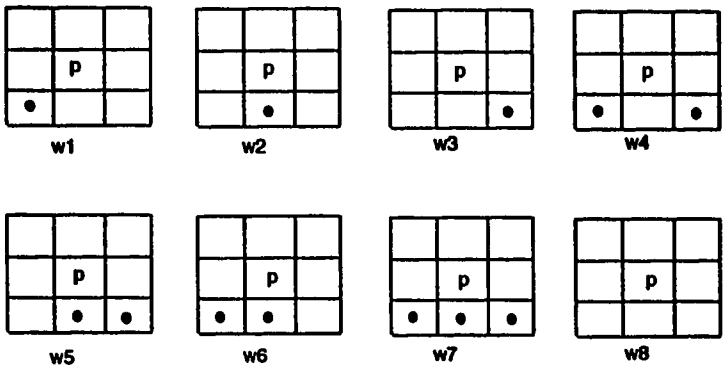


Figure 4.6: Possible configurations of window (d) in Figure 4.5. From[NS84]

FALSE otherwise. In order to check for right edge, top edge, and bottom edge points, similar conditions have to be tested.

for right safe point:

$$S_0 = n_4 \cdot (n_5 + n_6 + n_2 + n_3) \cdot (n_6 + \bar{n}_7) \cdot (n_2 + \bar{n}_1)$$

for top safe points:

$$S_2 = n_6 \cdot (n_7 + n_0 + n_4 + n_5) \cdot (n_0 + \bar{n}_1) \cdot (n_4 + \bar{n}_3)$$

for bottom safe:

$$S_6 = n_2 \cdot (n_3 + n_4 + n_0 + n_1) \cdot (n_4 + \bar{n}_5) \cdot (n_0 + \bar{n}_7)$$[NS84]

The high level description of the algorithm is presented in Figure 4.7.

Figure 4.7: SPTA: Layout of the Algorithm

Generality of the Neocognitron

Our system is designed so that it can handle characters having strokes up to *three* pixels in width, although the training was conducted on patterns of one pixel width only. In one experiment we took images of 264 handwritten characters. The average stroke widths of these characters was 2.5 pixels. The system was able to recognize 239 characters correctly. The is equivalent to 90.5% recognition accuracy. We have similar results when these characters were thinned. When these characters were thinned, 240 characters were recognized.

This observation can be explained by the role of the first S-layer. As will be shown in Section 4.3.1, S-layer$_1$ detects lines of 12 different orientations. It has 12 planes to detect these orientations. Every plane was trained using a 3×3 training pattern containing exactly three elements. The selectivity controller, r, was assigned a value of "1.7". This is to allow every plane to detect a pattern with one additive and one subtractive element in a 3×3 window. In addition, this value allows every plane to tolerate up to three additive elements and no subtractive one. It means that these planes can act as edge detectors.

Figure 4.8 explains the role of the first layer in minimizing the effect of stroke

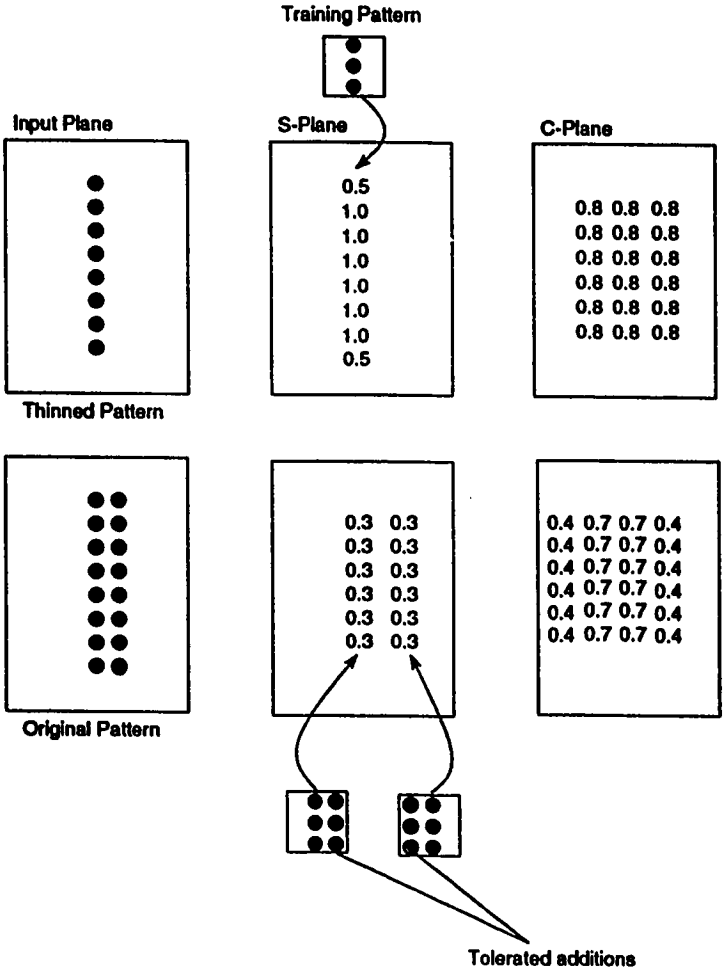


Figure 4.8: Role of the first layer in handling patterns with varying stroke widths

width variance. The Figure summarizes an experiment where we fed the network with a pattern before and after thinning it. When the pattern was thinned, the training pattern correlated well with many parts of the input pattern. This can be identified by the presence of the value 1.0 in the S-plane. When the pattern was fed without thinning, S-plane used the same training pattern to detect edges of the input pattern. Since the training pattern does not match with any part of the input pattern, we can not expect to have very large values in the S-plane. In both cases, the C-planes smooth the difference by blurring the values of the preceding S-planes. As a result, the same C-plane in both cases, will produce outputs that are nearly equivalent. In addition this slight difference between the two outputs can be tolerated by the succeeding S-layer. Therefore, the effect of varying stroke widths is absorbed mostly with the aid of the first layer, and partly by the 2nd S-layer, provided that proper parameters are chosen.

This does not mean that this argument can be extended and applied to characters with very thick strokes. Suppose that the system is to process characters with strokes of four pixels wide. The output of the C-plane when the character is not thinned will be quite different than in the case when thinning is used. This difference can not be tolerated by the succeeding S-layer, which would result in rejection of the thinned character. After testing on several stroke widths, we found that our system is not able to recognize unthinned characters with strokes more than four pixels in

width.

As a conclusion for this discussion it would be clear that although the system is robust to varying stroke widths, thinning is still used to account for cases where the characters have very thick strokes.

4.2.3 Segmentation

Character Recognition is composed of the following phases:

1. observation
2. segmentation
3. classification

A general character classifier would follow these steps in the mentioned order. However, knowledge of the context in which writing is conducted can help in the recognition phase. In addition, knowledge of any constraints confining the number of characters, their inter-relations (as in zip-codes), or the set from which they are drawn would be of good help in the recognition phase [FNK92]. The goal of the segmentation phase is to segment the image and pass a character per segment to the recognizer.

Segmentation becomes a major obstacle in the recognition of touching characters. In free handwriting, characters are expected to touch at different positions. Segmentation in the context of handwriting suffers from the following problems [Bok92]:

- A segment may contain multiple characters or part of a character
- A segment may contain both character and noise
- A segment may contain nontext (graphics, lines, ...etc)

Our program is confined to the recognition of handwritten Hindi numerals. When writing the numbers even when handwritten we do not expect them to touch. The segmentation method that we used is presented in [KPB87], and it is based on the idea of the *vertical cut*. At every horizontal position, pixels along the vertical line passing through this position are counted. Horizontal locations with very low values represent the corresponding cuts in the image.

4.3 Training Set

The neocognitron is a general architecture that can be applied to a variety of pattern recognition systems. Once the network is built, it can be used to recognize different

sets of patterns. Two jobs need to be done for the network to be targeted toward a particular set. The parameters have to be selected correctly, and the training set has to be established. These two jobs go usually in parallel. In our implementation, a very large time was spent designing the training set and selecting the correct parameters.

Our network consists of four layers in addition to the input layer. Every layer is composed of an S-sublayer and a C-sublayer. During training, every layer has its own training set. The number of training patterns in every layer is equal to the number of planes in that layer. Since supervised training is used, every pattern needs to be presented only once. A plane is normally trained on several patterns of a particular feature. This is necessary in order to make the plane more tolerant to deformations of that feature. In a case where large deformations need to be tolerated, training one plane on various variations of that feature might result in that plane not detecting any of them. In this case, two or more planes are required. Every plane is trained on patterns that are similar, and the outputs of these planes are joined in one C-plane, as will be show in detail.

In order to provide better alignment and centering, dimensions of the planes and the training patterns are chosen to be odd integers. The value of the fixed connection $c(v)$ is large at the center and decreases at the periphery. As a result, the variable excitatory connections are reinforced more strongly at the center. There-

fore, training patterns of a particular plane are required to align at their centers as precisely as possible.

Training patterns are presented one by one to the network. Every pattern is centered in the input plane. The output of layers 1 through $i-1$ (The layer preceding the one undergoing training) are computed. The connections leading to layer i are modified according to the training equations presented above.

The network is designed so that every plane has its own selectivity controller r. However, usually most of the planes in a layer share the same value of r. Some planes might need to have different selectivities than others, thus the provision of one selectivity controller per plane.

4.3.1 Training Patterns for Layer U_{S1}

The first layer is designed to detect lines of different orientations. The S-cells of this layer have receptive fields that are 3×3 in size. This is because the training patterns of this layer are of this size. Training patterns of this size can only detect four line orientations shown in Figure 4.9.

The ability of the first layer to detect slanted lines will improve if it could further be trained on lines with 1:2 slanting ratio. Figure 4.10 shows the complete training

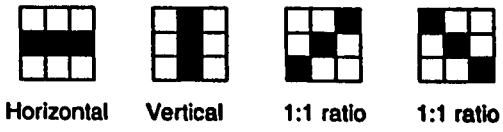


Figure 4.9: Possible orientations detected by 3 × 3 training patterns

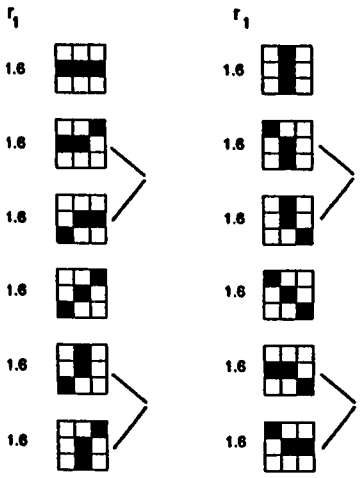


Figure 4.10: Training patterns for U_{S1}. The lines show the joining of some S-cells outputs. From [FW91]

set for layer 1. Each pair of lines shows the joining of the outputs of two S-cells into one C-cell. Every pair of joined outputs detects a line with a 1:2 slanting ratio in a particular orientation.

4.3.2 Training Patterns for Layer U_{S2}

The second layer is designed to detect local features of all the characters. Therefore, it is expected to have the largest number of training patterns. Training patterns of

this layer have 9×9 sizes. Layer 2 is very important for the training process. If the training patterns of this layer are perfectly designed, the network acquires excellent recognition power. Therefore, care must be taken in selecting the features needed to discriminate. In addition, alignment of training patterns should done accurately. Figure 4.11 shows the training patterns used to train the second layer.

Since the purpose of the network is to detect handwritten numerals, we expect that local features have large variances. For example, the top part of the hindi character '8' can have many variations. The last group of training patterns in the third column shows six possible variations of this feature, corresponding to six S-planes. These training patterns are supposed to cover all possible ways of writing this feature. Every row of this group represents a set of training patterns for one S-plane. Every row contains similar training patterns of a particular variation of this feature. The outputs of these six S-planes are joined into one C-plane. This C-plane is supposed to cover all of the variations of the feature under consideration.

An S-plane is usually trained on few training patterns in order to improve its ability to detect the possible variations. However, training one S-plane on all variations, will degrade the ability of this S-plane to detect these variations.

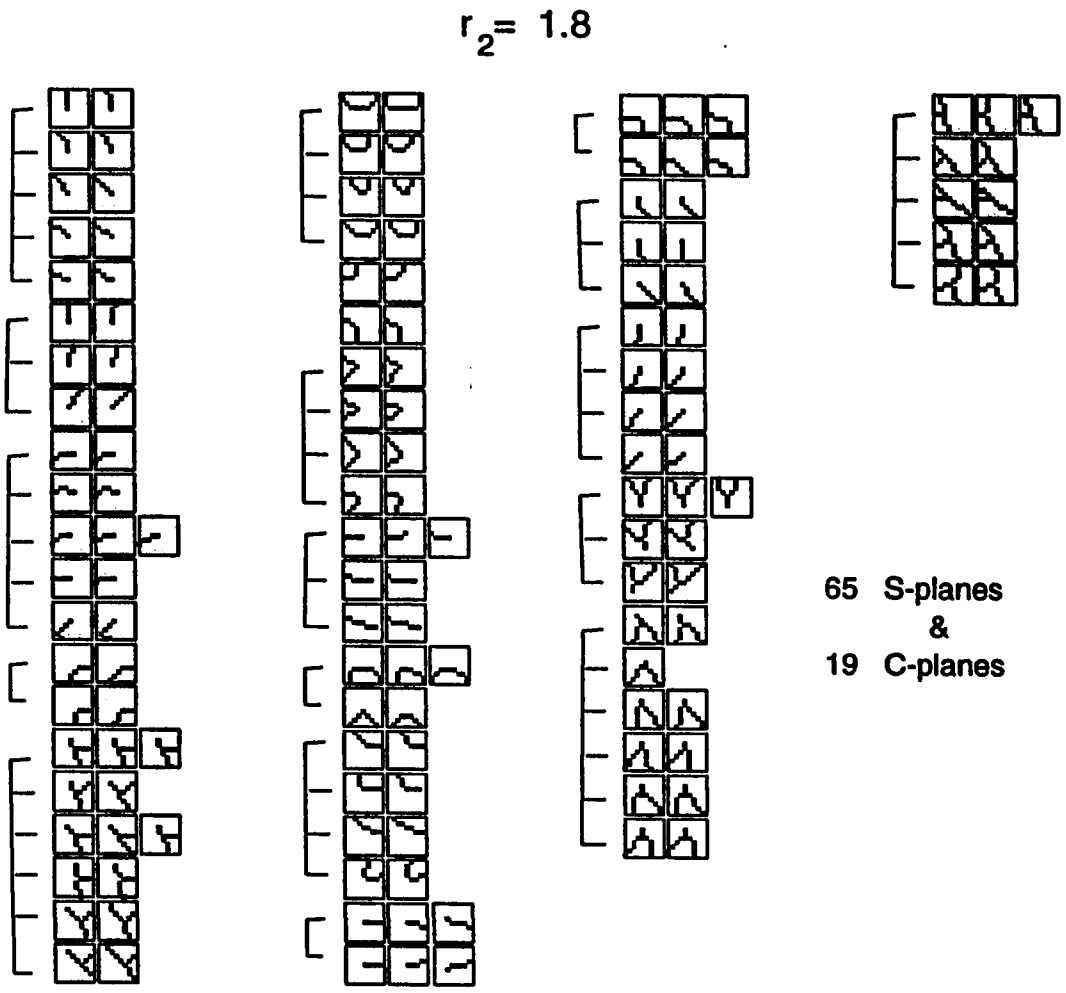


Figure 4.11: Training patterns used to train 65 cell planes of U_{S2}

4.3.3 Training Patterns for Layer U_{S3}

The third layer is designed to detect more global features. Figure 4.12 shows the training patterns used to train the 33 planes of U_{S3}. Some characters have large variations. Therefore, several S-planes are needed to detect all possible variations. Examples are characters 4 and 9. Generally in this layer, it is not necessary to present the layer with all the possible variations of a character. This is because large amounts of deformations have already been absorbed by the previous layer. This is why few patterns corresponding to every character are used to train the layer.

In this layer no outputs of S-planes are joined. This is because, as explained above, only one S-plane is needed to detect particular variations of a global feature of a character.

4.3.4 Training Patterns for Layer U_{S4}

The fourth layer is the recognition layer. The 33 S-planes are trained on the same patterns used to train layer U_{S3}, shown in Figure 4.13. However the goal of this layer is to consolidate all of the global features detected in the previous layer and decide on which character to be selected.

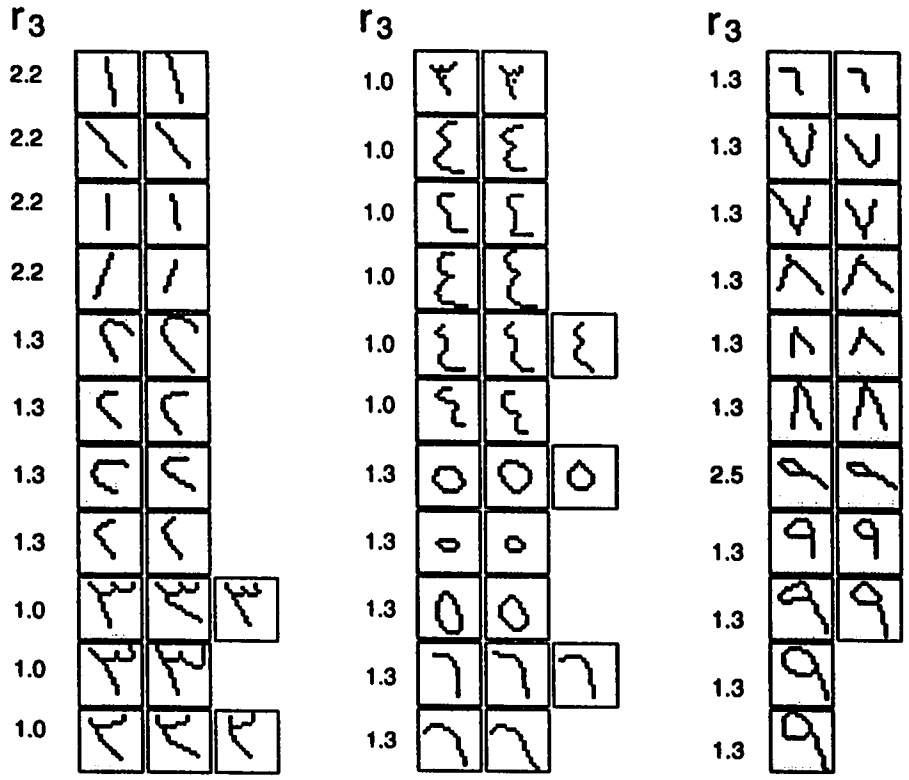


Figure 4.12: Training patterns used to train 33 S-planes of U_{S3}

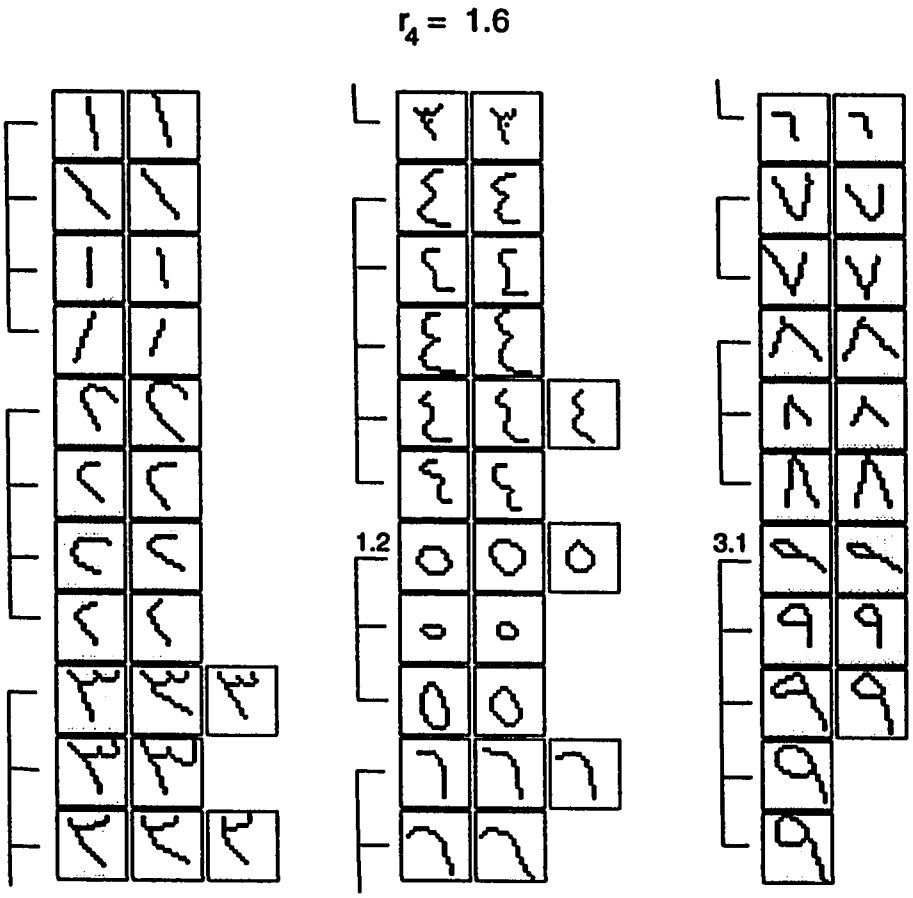


Figure 4.13: Training patterns used to train 33 S-planes of U_{S4}

As described in previous sections, the C-planes of the U_{C4} layer are 1×1 in size. The outputs of the C-cells in this layer undergo lateral inhibition. This means that, the outputs of the ten C-cells (corresponding to the ten numerals) are computed. If the output of one cell is larger than the average response of all the cells, this cell's output is retained. Otherwise, the cell does not respond.

4.4 Sample Data

The system has to operate on real data. In order to collect specimens, we asked 20 friends from the Information and Computer Science Department to use their handwritings to print on sheets of paper the complete set of Hindi numerals. In addition, we collected another specimens from 45 individuals at the telephone booths in two days. Twenty persons were asked to submit their handwritten sets in the morning of the first day. The remaining twenty five sets were recorded by different persons in the evening of the second day. The total number of collected characters is 650.

When performing the tasks of training the neocognitron and tuning its parameters, three different sets (A, B, and C) of characters were composed. The 650 collected characters were randomly split between sets A and B. Set A consists of 300 characters while set B consists of 350 characters. Characters pertaining to set

C were manually selected from all of the 650 collected characters. Therefore, there will be an overlap between set C on one side and sets A and B on the other side.

In order to accomplish the supervised training of the network, we need a training set. Set C serves the purpose of training the network. It contains training patterns for every character that the network is trained to recognize. When deciding on the training set, we made sure that the training patterns of a particular character cover all of the different variations of this character. The training patterns were selected from among the 650 collected characters. All 65 specimens pertaining to one character were grouped together. The training patterns were then manually selected from these specimens based on the variations of this character.

Table 4.1 lists the number of training patterns for every character taught to the network. We notice that the numbers are not the same. Some characters do not have large variations. As a result only small numbers of training patterns are used for training the network on these characters. Other characters have large variations. Examples are the Hindi '3' and '4', which have large variations when written by different people. These characters require large numbers of training patterns to cover all of the major variations that they can have.

In addition, there is no correlation between the patterns in the training set. Training patterns do not have to be drawn from the same individuals. Every char-

Character	Number of training patterns
1	8
2	8
3	10
4	11
5	7
6	7
7	4
8	6
9	8

Table 4.1: Number of training patterns for every character in the training set

acter was assigned a subset of training patterns based on the possible variations of this character.

A tuning set is needed to tune different parameters of the system to their optimal values. Either of sets A and B can serve the purpose of tuning the parameters. We decided to use set A which contains 300 characters. The process of tuning the system goes as follows. For every parameter assignment, the training set was used to update the weights of the network, in a way train it. The tuning set is then used to measure the performance of the system for that particular parameter selection. This process is repeated for all the attempted parameter assignments. Once a sufficient number of candidate parameter assignments are attempted, the one that resulted in the optimal performance is finally decided on.

In order to check if the system is robust, we have to test it on characters that were

Parameter	Range
R1	1.7
R2	1.0 - 2.0; incr = 0.1
R3	1.4 - 2.0; incr = 0.1
R4	0.8 - 2.0; incr = 0.1

Table 4.2: Ranges of the selectivity controllers used in a small experiment

never used in tuning the network. As might be expected, either of set A or B can be used for testing the system. Since we selected set A for tuning the parameters, set B will then be used for testing the final performance.

4.5 Analysis of Selectivity Controllers

In order to determine the optimum values for r's, the selectivity controllers, we ran the program on 140 characters using a reduced training set in order to save time. We experimented on all combinations of values of r's from the ranges shown in table 4.2.

After conducting this experiment, we found that the network had optimum results when the following parameters assignment was chosen:

$$r1 = 1.7;\ r2 = 1.6;\ r3 = 1.2;\ r4 = 1.8$$

In order to check if these assignments will also produce optimal results when considering the normal training set, additional experiments have to be conducted. In the forthcoming analysis, we tested the network on 300 characters using the complete training set. In analyzing every parameter, only this parameter was changed. The remaining parameters were fixed.

4.5.1 Selection of r for U_{S1}

Layer U_{S1} is responsible for detecting lines and edges of different orientations. Figure 4.14 plots the performance of the system with respect to different values for $r1$. The X-axis or the horizontal axis shows different values of $r1$ and the other axis shows the performance as a percentage. Four measures are plotted in the Figure, namely:

The recognition rate : the most expressive measure and it is normally used to characterize the performance of the network.

The rejection rate : expresses the percentage of characters that the network could not classify

The confusion rate : denotes the percentage of cases in which the network reported that the processed character belongs to two or more classes

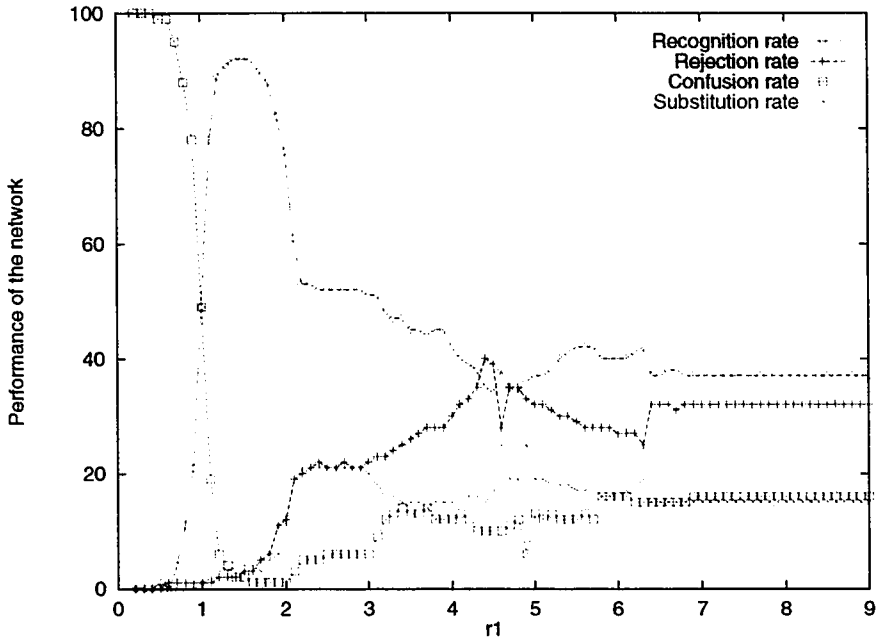


Figure 4.14: Performance of the system with respect to $r1$

The substitution rate : expresses the percentage of characters that were determined to belong to a wrong category. Normally, this measure is used to report the error rate of the network.

From the Figure we can see that the network has the optimum recognition rate when $r1$ is assigned any of the values: 1.4,1.5,1.6. At these values, all of the other three measures have very low values not exceeding 4%. Table 4.3 displays the rates of the various measures when $r1$=1.6.

The confusion rate is very high for values of $r1$ less than 1.0, while the other measures have low values. This behavior is expected recalling the fact that layer U_{S1}

Measure	Rate %
Recognition	92
Rejection	3
Confusion	2
Substitution	3

Table 4.3: Performance measures values when $r1 = 1.6$

works as a line orientations detector. The smaller the value of $r1$ gets, the smaller the selectivity of U_{S1} gets . In this way a plane in U_{S1} whose receptive field is 3×3 will tolerate more additive bits. As a result, a plane which is supposed to detect one line orientation only, will be detecting many orientations. Equivalently, a line orientation, which is supposed to be detected by one S-plane, will be detected by many S-planes, if not by all of them.

Let's consider the second S-layer; namely U_{S2}, and extending the former discussion we have the following situation. Upon presenting a local feature, intended to train a plane in U_{S2}, nearly all of the planes in U_{S1} will respond. This will be the case for almost all of the training patterns of U_{S2}. Therefore, the excitatory weights developed by a particular plane in U_{S2} will resemble weights developed by other planes. As a result, every plane in U_{S2} will respond upon presentation of any local feature. This argument can be extended to the third and forth layers. We will have a situation where most of the C-planes in U_{C4} will respond upon presenting any character, resulting in such a very high confusion rate.

From the Figure it is also clear that for values of $r1$ greater than 6.4, all of the measures will have steady values. Definition of the selectivity of S-cells of the first layer will help understand this behavior:

$$s = \frac{1 - \frac{n_0}{N}}{\sqrt{1 + \frac{n_1 - n_0}{N}}} \tag{4.21}$$

where n_0 is the number of elements which are active in the training pattern $P(v)$ but have zero values in the input pattern $p(v)$. n_1 is the number of elements which are active in $p(v)$ but have zero values in $P(v)$. N is the number of active elements in $P(v)$. All features of this layer have three elements, therefore N will have the value 3. When $r1$ satisfies the condition : $s > \frac{r}{1+r}$, the corresponding S-cell starts responding. Substituting the value of s in this condition we have :

$$\frac{1 - \frac{n_0}{N}}{\sqrt{1 + \frac{n_1 - n_0}{N}}} > \frac{r}{1 + r} \tag{4.22}$$

We can see that the larger the value of $r1$ is, the larger the selectivity of the S-cell becomes. This translates into less tolerance to additive or subtractive elements. In the following example we will see the effect of different values of $r1$ on the tolerance of the S-cell. When $r1$ is assigned a value of 2, we will have the following equation:

$$\frac{1 - \frac{n_0}{3}}{\sqrt{1 + \frac{n_1 - n_0}{3}}} > \frac{2}{3} \tag{4.23}$$

By trying different values for n_0 and n_1, we find that the cell can tolerate the following modifications:

no deletions, one addition

no deletions, two additions

no deletions, three additions

one deletion, no additions

When $r1$ takes the value 4, by substituting into equation (4.22) and by trying different values of n_0 and n_1, we find that the cell can tolerate the following modifications:

no deletions, one addition

one deletion, no additions

When $r1$ takes the value 6.4, we notice that the cell can not tolerate any combination of additions and deletions. In other words, the S-cell will have the maximum selectivity, and any amount of noise or deformation can not be tolerated. Irrespective of how large $r1$ gets then, the selectivity stays maximum and does not change, thus the steady performance of the system.

4.5.2 Selection of r for U_{S2}

The second layer is responsible for extracting local features from the input characters. Recognition of characters depends to a large extent on the selection of local

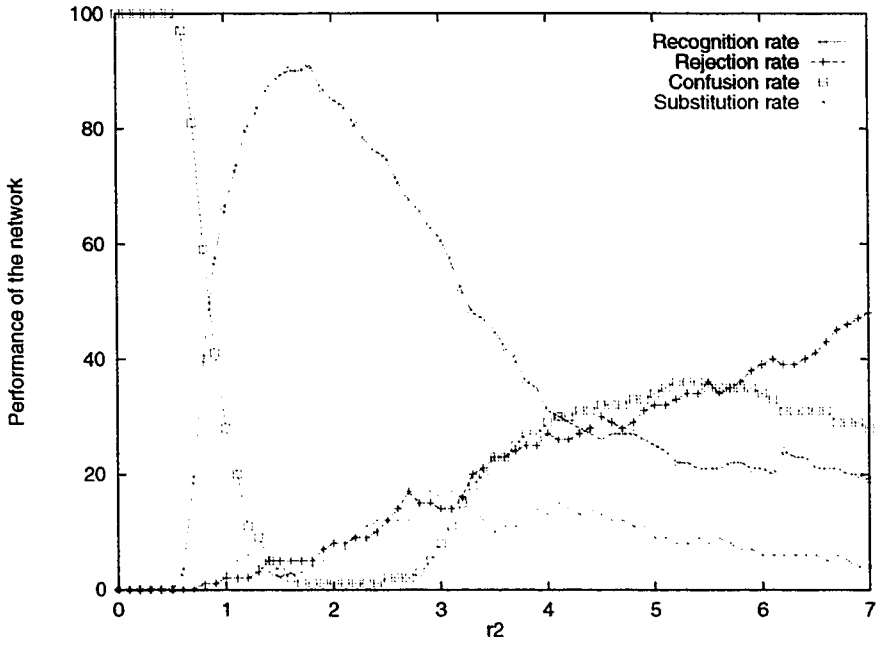


Figure 4.15: Performance of the system with respect to $r2$

features. This is why the behavior of this layer is unique and different from other layers.

Figure 4.15 plots the curves of different measures with respect to different values of $r2$. When $r2 = 1.8$, the recognition rate has a maximum value of 91 %. Values of different measures for $r2 = 1.8$ are presented in table 4.4.

Measure	Rate %
Recognition	91
Rejection	5
Confusion	1
Substitution	3

Table 4.4: Performance measures values when $r2 = 1.8$

We can see from the Figure that at very low values of $r2$ (less than 0.5), the confusion rate is very high. It decreases until $r2$ becomes 1.8, at which the confusion rate is minimum. As $r2$ increases beyond 1.8, the confusion rate starts mounting up until $r2$ reaches 5.3, after which it starts to decrease until it fades.

Such a strange behavior is expected recalling that a character is composed of, or can be described by a set of local features. In order to achieve optimum recognition, every character must have a unique or distinguishing set of local features. When deciding on the set of local features, two factors are considered:

1. The set of local features must fully describe the character

2. A maximum disjoint should occur between sets of different characters

The set of features designed for this layer was carefully selected, therefore the remaining task is to fine tune $r2$ to satisfy both conditions.

In order to understand the effect of local features on the behavior of different curves, we ran an experiment where we evaluated the performance of the network for different values of $r2$. Table 4.5 presents the average number of local features detected for every character with respect to different values of $r2$.

When $r2$ is very small (less than 0.5), the selectivity of the S-cells is very low, and as a result a lot of S-planes in U_{S2} will respond upon presenting any character

$r2$	Avg. # features	Confusion rate %
0.5	12.84	100
1.8	3.91	1
4.0	2.68	29
5.3	1.89	36
7.0	0.97	29
10.0	0.29	16
20.0	0.02	0

Table 4.5: Avg. number of features for different values of $r2$

during *training* as well as during *operation*. This means that every character will have a very large set of local features representing it, violating condition 2 above. In fact when $r2 = 0.5$, nearly every character will trigger 13 features out of the total 16 features. A lot of overlap will be between sets of different characters and hence we have a very high confusion rate.

As $r2$ increases, so does the selectivity of the S-cells. The set of local features describing a particular character will get smaller and more distinguishing. As a result, confusion starts to decrease. When $r2$ reaches 1.8, every character will have a fully representing and very distinguishing and minimal set of local features. Infact, at this value it was found that on average four features will be extracted from every character (see table 4.5). When designing the set of local features for this layer we found that every character on average is associated with 3.78 features. This observation agrees with the result of the experiment. This explains why when $r2 =$ 1.8 confusion is minimum while recognition is maximum.

As $r2$ increases above 1.8, some local features will start to fade or have their planes' outputs decrease. A character now is not fully described by the second layer. Some pieces of information that are needed to represent that character will start to be missing, violating condition 1 above. As a result characters will start to confuse resulting in an increase in the confusion rate.

The confusion rate keeps increasing until $r2$ becomes 5.3 at which confusion is maximum and begins to fall again. When $r2$ is 5.3, nearly one half of the features describing a character will be missing. This means that one half of the information needed to represent a character will not be extracted. One might be tempted to say that if a character is missing half of its features, then we must only expect rejection. This argument is true in case the character was fully represented during the training phase. However, since all selectivity controllers (including $r2$) maintain their values during training as well as during operation, confusion is expected. Therefore, because a character is missing half of its information at $r2 = 5.3$, confusion rate is very high.

As $r2$ increases above 5.3, more information will be missing from the character. For example, when $r2 = 7.0$ nearly 3/4 of the features will be missing from the set of local features of a character. Such a loss of information can not be tolerated by the network. As a result, as $r2$ exceeds 5.3 confusion starts to fade in favor of rejection (see table 4.5).

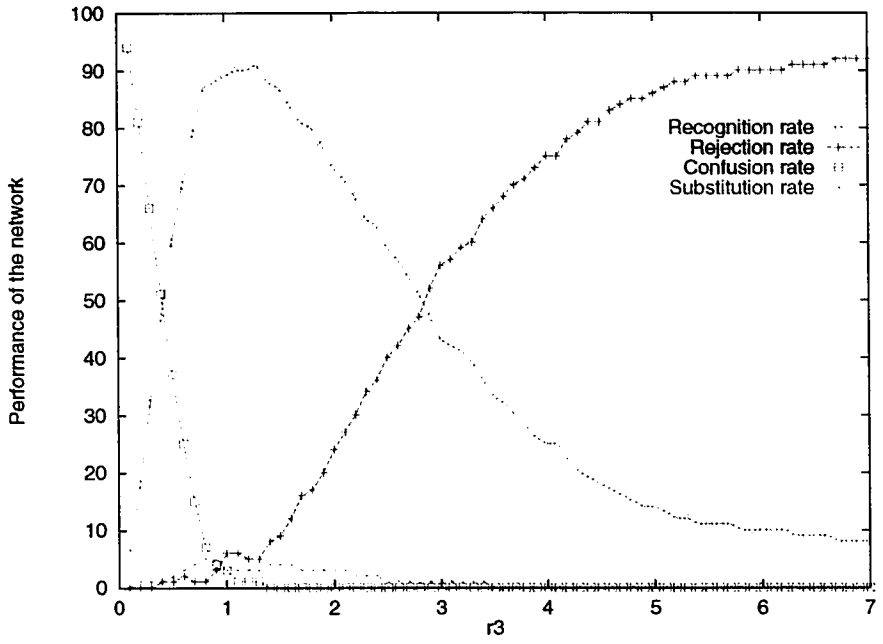


Figure 4.16: Performance of the system with respect to $r3$

4.5.3 Selection of r for U_{S3}

This layer is responsible for detecting more global features than the previous two. The maximum recognition rate achieved is 91% when $r3 = 1.3$. Figure 4.16 plots the different curves of the measures versus $r3$. Table 4.6 presents the values of the different measures for $r3 = 1.3$.

Measure	Rate %
Recognition	91
Rejection	5
Confusion	1
Substitution	3

Table 4.6: Performance measures values when $r3 = 1.3$

From the Figure, we see that the confusion rate starts very high and drops very quickly. At small values of $r3$, it is expected that the confusion rate is large. The same argument considering the confusion rate of the first and second layer for values of r less than 1.0 applies here. When $r3$ is smaller than 1.0, the selectivity of the planes is very low and as a result most of the planes of U_{S3} will respond upon presenting any character to the network. When $r3$ exceeds 1.0, the behavior is different from that of the second layer. As we know, the third layer is responsible for extracting *global* features whereas the second layer extracts local features.

When one of the local features of an input character goes undetected, it might happen that the network confuses between this character and some other character. Equivalently, the network might substitute this character for another one. This means that missing one local feature, either by its absence from the input character or by the inability of the second layer to detect it because of high selectivity, might not stop the network from responding either by confusion or substitution.

However, the network can not tolerate missing a global feature. Absence of a global feature means that a large portion of the character is not being detected. Considering the features of U_{S3}, it is not possible that removing a global feature will transform one character into another or create a confusion between two characters. As $r3$ exceeds 1.0, the selectivity of S-cells of U_{S3} increases. As a result, some global features will start to go undetected. Therefore, the confusion rate and the

substitution rate will start to diminish.

4.5.4 Selection of r for U_{S4}

This layer is the recognition layer. Figure 4.17 plots the different curves of the measures versus $r4$. The maximum achieved recognition rate is 92% when $r4 = 1.4, 1.5, or 1.6$. When $r4 = 1.6$, table 4.7 presents the different values of the different measures.

The S-cells of U_{S3} detect global features. Many times it is the case that the receptive fields of an S-cell would cover most of the character, if not all of it. Therefore, the roles of U_{S3} and U_{S4} are somehow similar. As a result, we would expect their curves to have similar behaviors.

However, the curves of U_{S4} are less steep. Considering the behavior of the neocognitron, we know that errors and deformations are not corrected at one layer only. Rather, errors and deformations are absorbed at every layer. Therefore, the amount of errors at the entry of U_{S4} is smaller than the amount of errors at the entry of U_{S3}. This means that characters will be corrected at entry of U_{S4} more than they were corrected at entry of U_{S3}. Therefore, changing $r4$ will have less effect on the performance measures than changing $r3$.

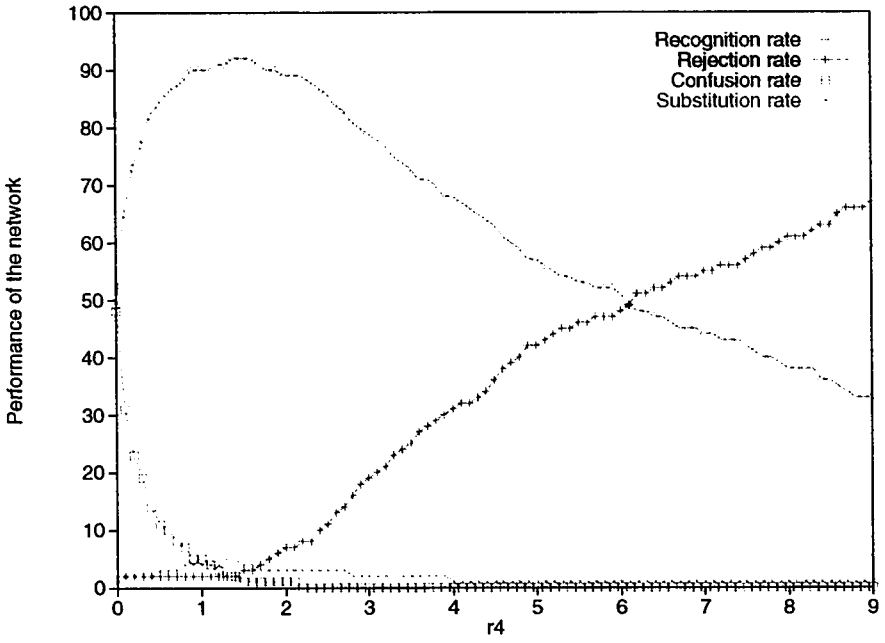


Figure 4.17: Performance of the system with respect to $r4$

Measure	Rate %
Recognition	92
Rejection	3
Confusion	1
Substitution	4

Table 4.7: Performance measures values when $r4 = 1.6$

After the former analyses were conducted, we chose to assign the selectivity parameters the following values:

$$r1 = 1.6, r2 = 1.8, r3 = 1.3, r4 = 1.6$$

Three values of $r1$, namely 1.4, 1.5, and 1.6, result in the optimum performance of the network. However, since a larger value of $r1$ results in a better orientation selectivity, we choose to assign $r1$ the value **1.6**.

4.6 Analysis of ALPHA

The parameter ALPHA, used in equation (4.6), is a positive constant that determines the degree of saturation of the C-cells. Fukushima did not stress any importance to specific values of this parameter, and in his research he used two values: *0.7* and *1.0* interchangeably. The equation for calculating the saturation factor is repeated here for clarity:

$$\psi[x] = \frac{\varphi[x]}{ALPHA + \varphi[x]}$$

The output of an S-cell falls within the range : 0..1. The output of a C-cell is computed according to the formula (4.8). Since $d_l(v)$ can not take values larger than one, and suppose that the C-cell's receptive field covers nine cells, then the

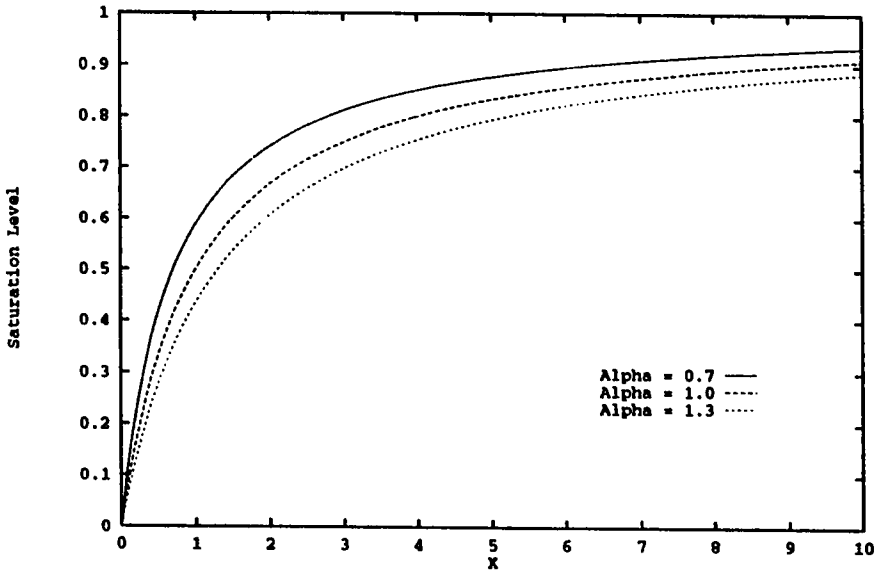


Figure 4.18: Saturation levels for different values of ALPHA

value computed by equation (4.8) can not exceed *9*. Figure 4.18 plots the saturation value versus x. Three curves each with a different value of ALPHA are plotted.

Suppose that we will do without invoking the saturation function, and suppose that the output of an S-cell is binary (0 or 1). The assumption of having binary output for the S-cell is not used in our implementation, but it will help in the understanding of the saturation function. Let's examine two cases: in the first case, only one S-cell responds within the receptive field of a C-cell, and in the second case all of the cells (say nine cells) will respond. If the saturation function is not invoked, then the output of the C-cell in the second case will be nine times the output of the cell in the first case. We have a very large difference although the C-cell detects the

same feature in both cases.

The saturation function is very convenient in this situation. It moderates the difference between both cases. At the same time, it does not completely eliminate this difference. In the example, the outputs of the C-cell after invoking the saturation function in both cases will be: *0.9* and *0.4* respectively.

In addition, the saturation function ensures that the output of a C-cell falls within the range: 0 to 1. Some implementations require that the C-cell responds with a binary output. In such a case if the saturation level is larger than *0.5* the cell responds, otherwise the output of the cell is inhibited.

A change in the value of ALPHA does not have a great effect on the performance of the network. Figure 4.19 plots the performance of the system with respect to ALPHA. We can see from the Figure that between 0.5 (best performance) and 3.0 the performance drops only by 9%. The performance stabilizes for values between 0.5 and 0.9, so we choose to assign ALPHA the value 0.7.

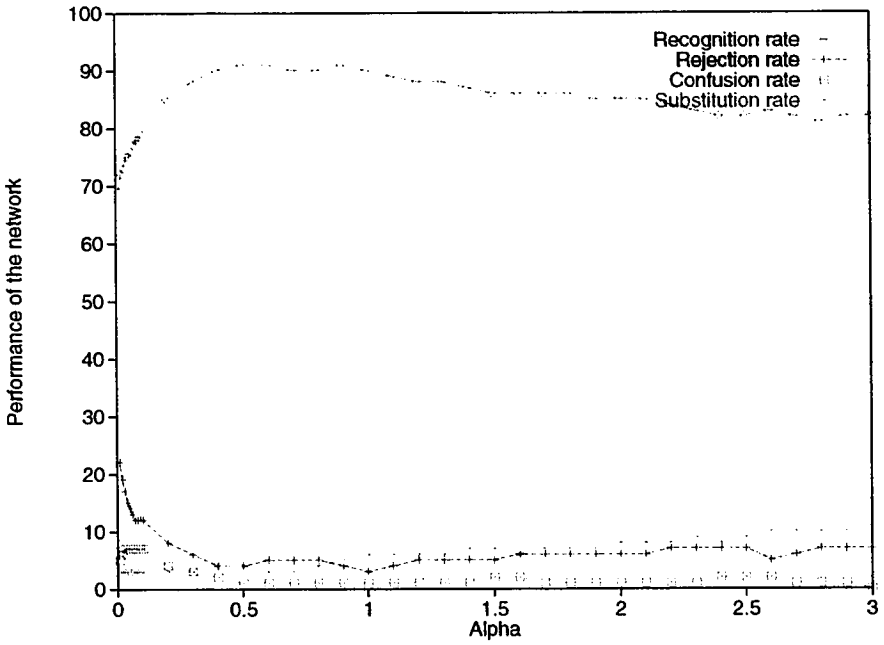


Figure 4.19: Performance versus ALPHA

Chapter 5

Results and Discussion

In this chapter a discussion of the need to process the Hindi zero separately is presented. Final results of performance are presented. A comparison in terms of error rate and recognition speed between our implementation and Fukushima's implementation is presented. Finally, performance of the system in noisy envronment is analyzed.

5.1 Handling Zero

The Hindi zero poses a problem in character recognition. One thinks of this character as a dot. However, this character takes many different shapes depending on different

handwritings. Figure 5.1 displays examples of different handwritten hindi zero's. Each pair of columns shows the characters before and after thinning. As we can see from the figure, the problem with the zero multiplies when the character gets thinned. The thinned character might take the shape of a stroke slanted in any direction. It might even become shapeless. Some thinning algorithms intended for use with arabic characters, thin the zero until it becomes a one pixel in size [Jam91]. However, in a noisy environment the thinned zero will not be segregated from noise.

In addition, a zero thinned to the shape of a stroke, might be easily confused with a short one, see Figure 5.1. Therefore, detecting the zero before it is thinned seems to be more effective.

Our decision was to detect the zero separately. Before the input characters are thinned, they are passed to a small network that tries to detect the zero's and filters them. The remaining characters are passed to the main network. Some zero's might not be detected by the zero filter. However, the percentage of detected zero's is still much higher than when they were thinned and passed to the main network for detection. In one experiment, we tested this filter on different handwritings of the zero. Out of *47* characters, only two zero's were unidentified, resulting in a *96%* accuracy rate.

The zero filter network needs a training set specific to this task. Figure 5.2 shows

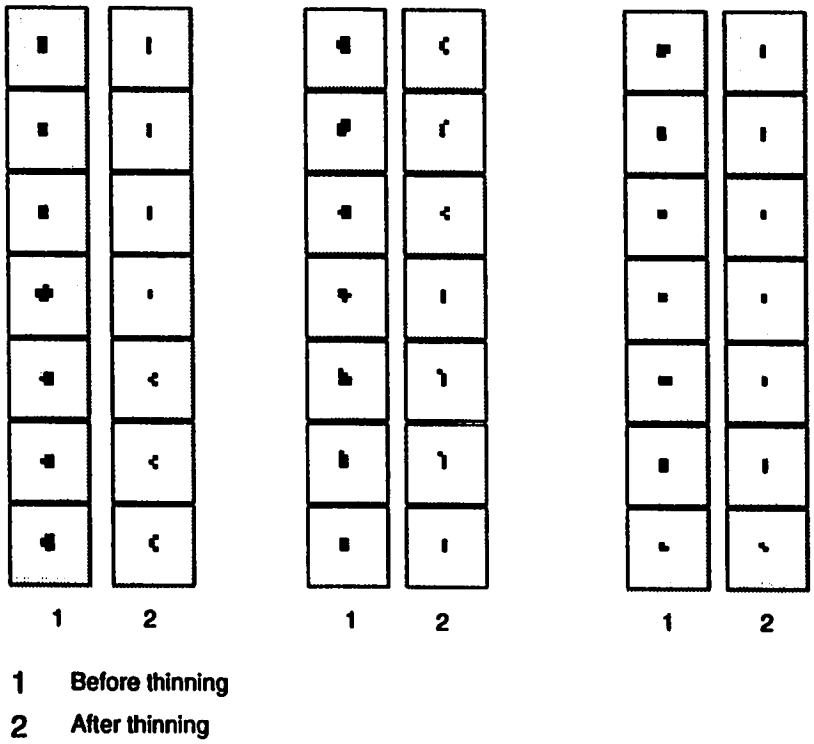
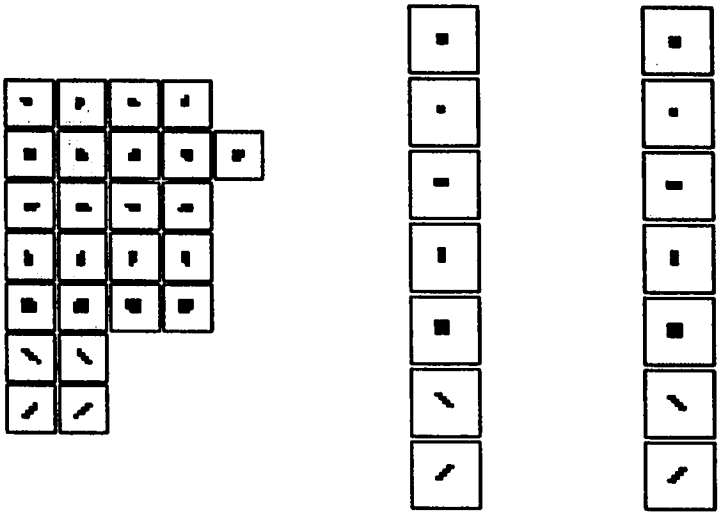


Figure 5.1: Examples of different handwritings of the hindi zero before and after thinning



DimSplane1 = *19x19*	FIELDS1 = *13*	Q1 = *1000*	BASE = *0.9*
DimCplane1 = *13x13*	FIELDC1 = *3*	Q2 = *1000*	ALPHA = *0.7*
DimSplane2 = *13x13*	FIELDS2 = *5*	Q3 = *1000*	
DimCplane2 = *9x9*	FIELDC2 = *5*		
DimSplane3 = *5x5*	FIELDS3 = *5*		
DimCplane3 = *1x1*	FIELDC3 = *5*		

Figure 5.2: Training patterns for the zero filter network

the training set intended for training the zero filter three layered network. The figure also displays the network's complete setup in terms of different parameters setting.

This strategy can be extended and applied to the rest of the characters. One can design small networks, each intended to detect one character only. This network will be trained on local and global features of this character. However, several design and implementation problems will arise if such strategy was conducted. Suppose

that two characters are close in shape to each other. This means that in the first few layers, many training patterns will be repeated in both networks. Since each network is separate, these patterns can not be shared; they have to be duplicated. As a result, the system will have an aggregate training set that is much larger than in the case of one large network.

In addition, every network has its own setup with the associated set of parameters. Therefore, one has to set all of the parameters of every network properly in order to achieve an optimum performance. If the intention is to detect ten numerals, as in our case, then the number of parameters that the designer has to worry about is ten times as large as that of one large network. These problems translate into a very large time for the network to train and for it to operate.

For all of these reasons, we preferred to use one large network. The exception was the character zero, which had to be passed to a separate network for the reasons explained above.

5.2 Results

In the previous chapter complete analyses were carried out for every parameter. The analyses were conducted on the tuning set consisting of 300 characters. Table 5.1

Measure	Percentage
Recognized	92%
Rejected	2%
Confused	1%
Substituted	5%

Table 5.1: Performance using parameters new assignments

summarizes the performance of the network using the selected parameters.

However, we would like to trade substitution for rejection . In addition, Some forms of characters like '2','6', and '8' were being confused with '1'. Besides, many forms of the '4' ,especially when written with a long slanting stroke, were either substituted or confused for a '9'. We also experienced difficulties in the recognition of character '3'.

The solution to these problems was to modify the selectivity controllers. The selectivity controller for '1' in the third layer was increased. We found that the confusion between '4' and '9' occurred because of the 29'th training pattern in Figures 4.12 and 4.13. Eliminating this training pattern would solve this confusion problem, however it would degrade the ability of the network to detect similar forms of '9'. As a compromise, we preferred to increase the selectivity controller of this plane in both layers. In addition, we had to decrease the selectivity controllers of planes detecting characters '3' and '4'. We found that it is only necessary to modify the selectivity controllers of the third layer. The selectivity controllers of U_{S4} were

Measure	Percentage
Right	95%
Rejected	1%
Confused	1%
Substituted	3%

Table 5.2: Performance using parameters final assignments

not modified. The only exception is the plane detecting the form of character '9' that is creating the trouble. Figure 4.12 shows the selectivity controllers of all the planes in the third layer. Figure 4.13 shows that only one plane pertaining to the group of the character '9' had its selectivity controller changed. After these modifications were incorporated, the results obtained are summarized in table 5.2.

The improvement that we had was mainly due to eliminating the confusion of some characters with '1', and the confusion between '4' and '9'. Moreover, planes detecting character '3' now have better selectivity and discrimination power.

We have to be sure that the system retains its good performance when new data are presented. In order to accomplish this task we tested the performance of the network on the testing set consisting of 350 characters. Table 5.3 summarizes the performance of the system. The results obtained are close to the results obtained from the tuning set. Therefore, our system is robust and general, and it can cope with a large variety of handwritten hindi characters. Figure 5.3 shows examples of correctly recognized handwritten Hindi numerals.

Measure	Percentage
Right	93.4%
Rejected	3.7%
Confused	0.6%
Substituted	2.3%

Table 5.3: Performance using new data

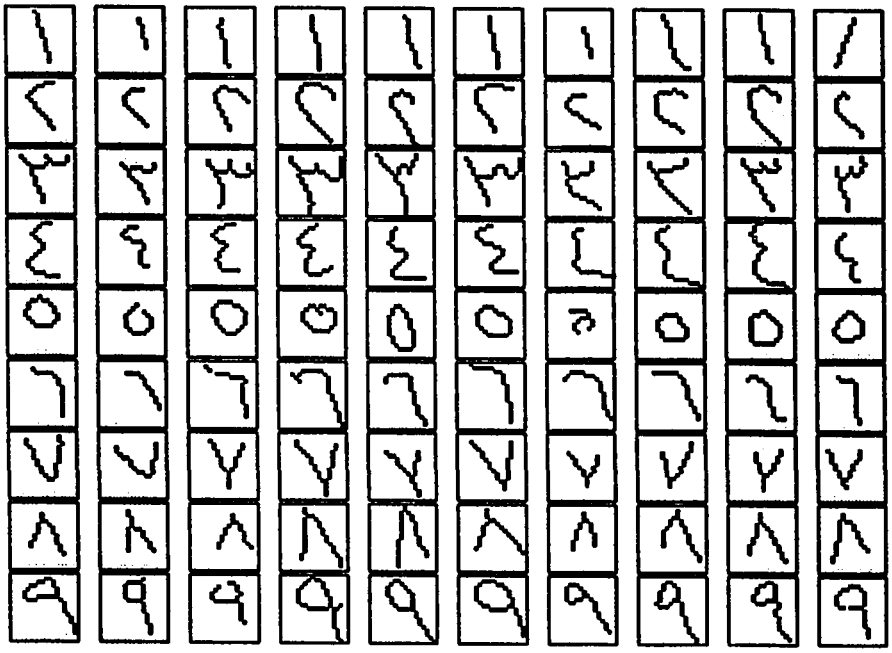


Figure 5.3: Examples of correctly recognized characters

5.3 Comparison with Fukushima

Fukushima has been publishing papers regarding the Neocognitron for the last 15 years. His earliest work concentrated on recognition of a set of the arabic numerals [FM82]. The unsupervised technique was used in his implementation. However, it was found to be slow, and the resulting Neocognitron was not robust. From 1983 and onward, he concentrated on the supervised technique, and he used it for the recognition of the complete set of arabic numerals [FMi83] [Fuk88a], and the entire english alphabet [FW91].

The optimum performance reported by Fukushima was a 16% error rate when considering the Arabic numerals. In his latest work [FOH94], Fukushima modifies the architecture of the Neocognitron. A dual C-cell layer is substituted for the normal C-layer. The error rate drops to 9%. Table 5.4 reports a comparison of the error rates between our system and that of Fukushima. Assuming that the error rate includes the rejection, the confusion, and the substitution rates together, our system still achieves a much better performance than that of Fukushima, as shown in the table.

Regarding the speed of the recognition time of the network, Fukushima reports that it took the network on average 1.5 seconds to recognize one character when running on a SUN SPARCstation[FW91]. After collecting statistics about the run-

Our system all measures added (Hindi numbers)	Fukushima (Arabic numbers)	Fukushima with dual C-cell (Arabic numbers)
5%	16%	9%

Table 5.4: Comparison of error rates

Layer	Number of Planes		Size of Planes		Size of Rec. fields	
	S	C	S	C	S	C
1	12	10	19x19	21x21	3x3	3x3
2	65	19	19x19	13x13	11x11	5x5
3	33	33	13x13	7x7	11x11	5x5
4	33	9	3x3	1x1	5x5	3x3

Table 5.5: Parameters affecting the recognition time of our implementation

ning times of our system, we found that recognition of a character consumes six seconds when running on the SUN SPARCstation.

Such an increase in time is expected considering that the structure of the neocognitron would be different when applied to different sets of patterns. The three parameters that affect the recognition time of the neocognitron are: number of planes, plane sizes, and receptive field sizes. Table 5.5 summarizes the different values of these parameters for every layer in our implementation. Table 5.6 summarizes these values for the implementation of Fukushima.

Computing the response of the network involves calculating outputs of S and

Layer	Number of Planes		Size of Planes		Size of Rec. fields	
	S	C	S	C	S	C
1	12	10	19x19	21x21	3x3	3x3
2	38	19	21x21	13x13	5x5	7x7
3	35	23	13x13	7x7	5x5	5x5
4	11	10	3x3	1x1	5x5	3x3

Table 5.6: Parameters affecting the recognition time of Fukushima's implementation

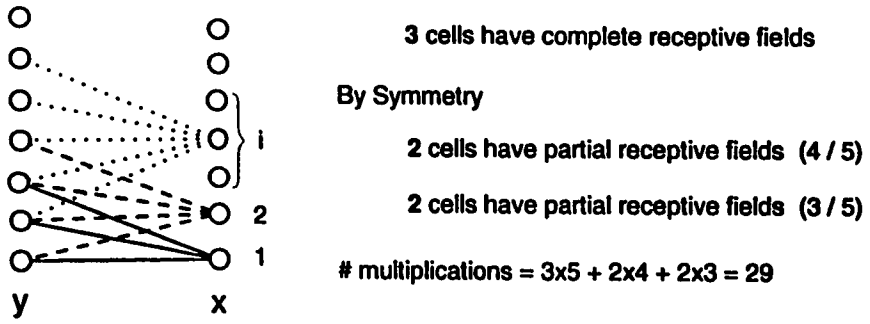


Figure 5.4: Complete and partial receptive fields

C cells for all layers. The dominating operation in these calculations is the multiplication operation. This is because, calculating the output of a cell in layer i involves computing the vector product of its receptive field with cells in layer $i-1$ that are covered by this receptive field. Therefore, knowledge of the number of multiplications would give an indication of the recognition time.

Its apparent that not all cells will have a complete receptive field. For example, cells at the boundary of a plane might have partial receptive fields. Figure 5.4 displays an example of two planes x and y, with cells of y being covered by receptive fields of cells of x. As the plane shows, three cells of plane x have complete receptive

fields, two cells have receptive fields each missing one cell, and two cells have receptive fields each missing two cells. Let m be the number of cells of plane x having complete receptive fields. Then, m can be defined as:

$$m = min(B_x, B_y - A + 1), \tag{5.1}$$

where B_x and B_y are the sizes of plane x and plane y, respectivley. A is the size of the receptive field. Number of multiplications (α) involving these cells is defined as:

$$\alpha = m \times A. \tag{5.2}$$

Number of multiplications (β) involving the remaining cells is defined as follows:

$$\beta = 2 \times (A-1) + 2 \times (A-2) + ... + 2 \times (A - \frac{B_x - m}{2}). \tag{5.3}$$

Putting these equations together, total number of multiplications is defined as:

$$M = m \times A + \sum_{j=1}^{\frac{B_x - m}{2}} 2 \times (A - j). \tag{5.4}$$

When considering the case of two dimensional receptive fields, the total number of multiplications will be M^2.

Applying this discussion to our layered architecture of S and C sublayers, we get the following total number of multiplications over all the network:

$$M = \sum_{i=1}^{N} [P_S(i) \times [\alpha_S(i) + \beta_S(i)]^2 + P_C(i) \times [\alpha_C(i) + \beta_C(i)]^2], \tag{5.5}$$

where

$P_S(i)$ is number of S planes in layer i,

$P_C(i)$ is number of C planes in layer i,

N is the number of layers,

$$\alpha_S(i) = m_S(i) \times A_S(i), \tag{5.6}$$

$$\alpha_C(i) = m_C(i) \times A_C(i), \tag{5.7}$$

$$\beta_S(i) = \sum_{j=1}^{\frac{B_S(i)-m_S(i)}{2}} 2 \times (A_S(i) - j), \tag{5.8}$$

$$\beta_C(i) = \sum_{j=1}^{\frac{B_C(i)-m_C(i)}{2}} 2 \times (A_C(i) - j), \tag{5.9}$$

$$m_S(i) = min(B_S(i), B_C(i-1) - A_S(i) + 1), \tag{5.10}$$

$$m_C(i) = min(B_C(i), B_S(i) - A_C(i) + 1). \tag{5.11}$$

Equations (5.6) to (5.11) consider one dimension of the planes and the receptive fields. However, the squares in equation (5.5) account for the two dimensional computation.

Substituting the values of A_S, A_C, B_S, B_C, P_S, and P_C in equations (5.6) to (5.11), and eventually in equation (5.5), the total number of multiplications involved in recognition of one character in our implementation is:

$$\begin{aligned}
&12 \times 55 \times 55 + 10 \times 57 \times 57 + \\
&65 \times 189 \times 189 + 19 \times 65 \times 65 + \\
&33 \times 113 \times 113 + 33 \times 35 \times 35 + \\
&33 \times 15 \times 15 + 9 \times 3 \times 3 = 2940238
\end{aligned}$$

By comparison, the total number of multiplications involved in recognition of one character in the implementation of Fukushima is:

$$\begin{array}{llllll} 12 & \times 55 & \times 55 & +10 & \times 57 & \times 57 & + \\ 38 & \times 99 & \times 99 & +19 & \times 91 & \times 91 & + \\ 35 & \times 59 & \times 59 & +23 & \times 35 & \times 35 & + \\ 11 & \times 15 & \times 15 & +10 & \times 3 & \times 3 & = 751142 \end{array}$$

Comparing the two figures, we find that the number of mutiplication operations in our implementation is $\frac{2940238}{751142} = 3.9$ times that of Fukushima's implementation. This result agrees with the observation that the recognition time of our network is approximately four times that of Fukushima's network.

5.4 Performance of the Network Under Noisy Environment

It is important for the system to cope with a noisy environment. Data is captured using a camera, so noise is inevitable. In our implementation we did not include any noise filtering technique. Data is captured and passed to the different modules of the system. All the modules concerned do not make any attempt to remove noise. We made sure to select a line thinning algorithm that does not filter the noise, nor

does it get confused with the presence of it.

In order to test the performance of the network under noise, we implemented an algorithm for generation of noise in binary images. This algorithm is presented in [ZG91], and the details of it follows.

Let f represent a binary image where $f_i = 1$ if the pixel is an object pixel, otherwise $f_i = 0$. The *mean* and the *variance* of the image are defined as:

$$\bar{f} = \frac{1}{N} \sum i = 1^N f_i \tag{5.12}$$

, and

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (f_i - \bar{f})^2 \tag{5.13}$$

Let N_0 be the number of object pixels, the variance of the signal is defined as :

$$\begin{aligned} \sigma_s^2 &= \frac{1}{N} \left[\sum_{i=1}^{N_0} (1 - \bar{f})^2 + \sum_{i=1}^{N-N_0} \bar{f}^2 \right] \\ &= \frac{1}{N} \left[N_0 (1 - \bar{f}^2) + (N - N_0) \bar{f}^2 \right] \\ &= \frac{1}{N} \left[N_0 \left(1 - \frac{N_0}{N} \right)^2 + (N - N_0) \left(\frac{N_0}{N} \right)^2 \right] \\ &= \frac{N_0}{N} \left(1 - \frac{N_0}{N} \right) \end{aligned} \tag{5.14}$$

Let N_n represent the number of noise bits in the image, then similarly the variance of the noise signal in the image is equal to:

$$\sigma_n^2 = \frac{N_n}{N} \left(1 - \frac{N_n}{N} \right) \tag{5.15}$$

The *signal-to-noise ratio* is defined as:

$$SNR = 10 \cdot log_{10}\left(\frac{\sigma_s^2}{\sigma_n^2}\right) \tag{5.16}$$

By substituting the expressions of σ_s and σ_n into equation (5.16) and solving for N_n we get :

$$N_n = \frac{N}{2} \cdot (1 \mp B) \tag{5.17}$$

where

$$B = \sqrt{\left[1 - 4 \cdot 10^{-\frac{SNR}{10}} \cdot \frac{N_0}{N} \cdot \left(1 - \frac{N_0}{N}\right)\right]} \tag{5.18}$$

Zhou decides on which of the two values of N_n to choose based on the following condition :

$$N_n = \begin{cases} \frac{N}{2} \cdot (1 - B), & for \frac{N_0}{N} \leq \frac{1}{2}, \\ \frac{N}{2} \cdot (1 + B), & otherwise \end{cases} \tag{5.19}$$

After calculating the number of noise bits corresponding to a particular SNR ratio, the exact locations of these bits need to be known. A random uniform distribution is used to generate random values between 0 and N-1. Noise is inserted in a specific pixel location by reversing its contents (color). Figure 5.5 plots the various system measures as functions of SNR.

We can see from the Figure that the system retains excellent recognition rate as long as SNR stays higher than 5 dB. The performance of the system deteriorates

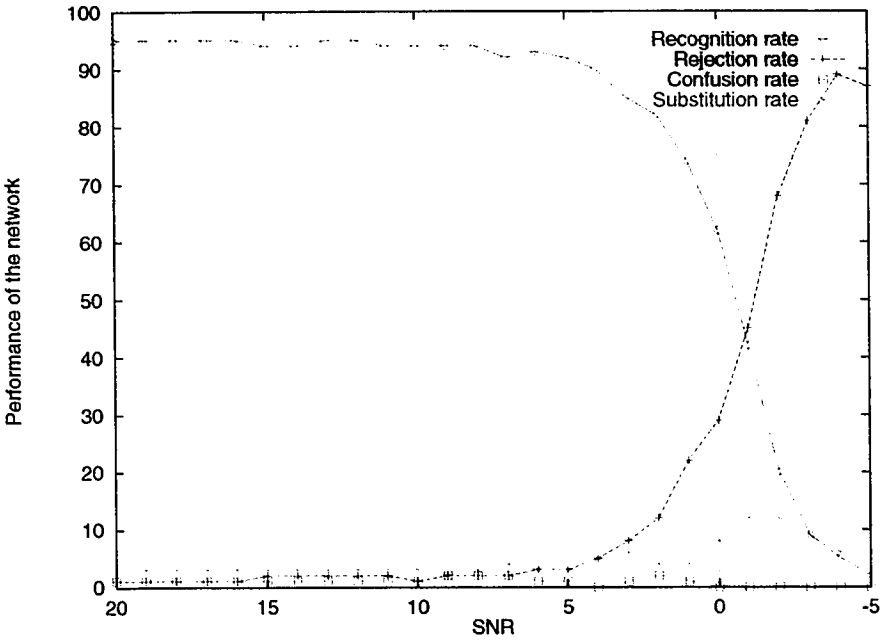


Figure 5.5: System performance under noise

when SNR gets lower than 0dB. In addition, it is clear that the rejection rate starts mounting up quickly as the recognition rate starts to decline. This property is required in good systems. It is demanded under very noisy conditions, that the system behaves in favor of rejection rather than confusion or substitution, which is the case in our system. However, having some substitutions is inevitable. This is because the noise signal by reversing bits of the image, might transform a character to a form that is much closer to another character. Therefore, we would expect to have substitutions especially at very low SNR's.

Coping with SNR's as low as 3 dB means that the system has a very high tolerance to noise. Table 5.7 summarizes some results obtained from this analysis.

SNR	Recognition Rate	Drop Percentage
20	95%	0
10	94%	1
5	92%	3
3	85%	10
0	62%	33
-5	2%	93

Table 5.7: Recognition rate as a of noise

Chapter 6

Conclusion

The objective of this thesis is to design a system for the recognition of the handwritten Hindi numerals. The targeted system should not be affected by translation or scaling of the numerals. In addition, it must tolerate minor deformations and minor rotations. The neocognitron, which is a neural network architecture, was designed and implemented in order to accomplish this task.

In our implementation, the C language was used for writing the code. Programs were compiled and run on a DEC Alpha (alif) machine running an OSF1 operating system. Only one program whose job is to digitize the gray level image was compiled and run on a PC running DOS version 5.0.

The network proved to have an excellent recognition power. A 95% recognition rate was achieved when testing the network on 700 characters. The network maintains excellent performance under noisy environments. Even at low SNR's, such as 3 SNR, the network maintains a recognition rate of not less than 85%.

In this chapter we conclude our work with a brief description of a digital neocognitron. Problems and limitations associated with the neocognitron are also discussed. Suggestions for future work are also presented.

6.1 The Digital Neocognitron

The neocognitron is a complex and very large network. It would be quite efficient to implement the neocognitron using VLSI techniques. This would allow parallel processing that well suit the functioning of the neocognitron and as a result would tremendously cut down the processing cost of both training and operating the neocognitron.

A digital neocognitron suitable for VLSI implementation is presented in [WE92]. The Digi-neocognitron (DNC) is derived from the normal neocognitron by a set of preprocessing approximations and definition of new model functions.

For example in digital circuit implementations, multiplications and divisions are

costly operations requiring large chip areas and long times. The DNC model uses single-term power-of-2 approximations. This is because multiplications and divisions by 2 is a shift operation with a very simple hardware implementation.

Bit widths are restricted. Weights are confined to 3 bit representations and neuron outputs to 7 bit representations. Complex functions such as square root are implemented by lookup tables.

A digi-neocognitron was implemented to recognize the five english characters *A* through *E*. The DNC has 5762 neurons with 181509 possible weights. The DNC has recognition rates similar to the normal neocognitron. However, in the case of DNC, the area-delay product factor is improved by two to three orders of magnitude [WE92].

6.2 Problems and Limitations of the Neocognitron

The neocognitron is a powerful neural network that can be used in a variety of pattern recognition applications. However, this architecture has some problems and limitations. The main two tasks involved in designing the neocognitron are:

- Selecting the proper parameters
- Deciding on a suitable training set

Selecting the proper parameters is not an easy task and a lot of experimentation is needed to reach near optimum assignments. The set of parameters that were experimented on include: four general selectivity controllers and Alpha [1]. The program offers the possibility of setting a different selectivity controller for every S-plane in every layer. Indeed, we had to further fine tune the system and set different selectivity controllers for some planes.

Deciding on a proper training set is also a time consuming task. Selecting the proper features to be extracted in the second layer was the dictating factor. In addition, changing the training set influences the first task. In fact, in many cases we had to go back and forth between the two tasks until proper parameter assignments and feature selections are reached.

As we discussed, the training set affects to a large degree the performance of the neocognitron. We found that adding more differentiating features or more examples of final characters to the training set would improve the performance provided that proper parameters are selected. However, this improvement does not come without a price. Adding more features involves creating more planes and interconnections.

[1] Used to compute the saturation factor of C cells

As a result, the training time and the operating time would increase accordingly. Therefore, a compromise between the running time and the performance has to be settled.

6.3 Future Work

6.3.1 Supervised vs. Unsupervised

Fukushima started his work on the neocognitron using unsupervised training. The last implementation using unsupervised training that he reported was in 1982. In [FM82] he reports an implementation of the neocognitron to recognize the five arabic numerals *1* through *5*. In the last decade, Fukushima published several articles concerning neocognitron implementations. In all of these implementations, supervised training was used.

Supervised training appeals to designers for it takes a much shorter time compared to unsupervised training. Besides, debugging (which involves tracking information flow) is much easier in the case of supervised training. This is because for every plane, the associated feature is known in advance.

However, supervised training confines the neocognitron to the recognition and

abstraction of the features that were used to train it. The selected features might not be the best ever one can design. Usually, one needs to spend a very long time modifying the training features until an acceptable level of performance is achieved.

A mechanism of combining supervised and unsupervised training seems to overcome the difficulties associated with using any of the two training techniques alone. This mechanism is used to train four layers of a hypothetical network:

Layer 1: Use the supervised technique to train this layer to recognize line segments with different orientations (As discussed in our implementation.)

Layer 2: Decide on a very certain set of distinguishing features (the number of features should be as small and precise as possible) and train a sufficient number of planes using the supervised technique. In order to complete the set of features for this layer, the unsupervised technique is used to train another set of planes from the same layer.

Layer 3: The unsupervised technique is used to train this layer.

Layer 4: Since this is the recognition layer, it seems reasonable to use the supervised technique to train it.

6.3.2 Neocognitron with Dual C-Cell Layers

In his latest publication, Fukushima presents a modification to the original architecture of the neocognitron [FOH94]. In normal neocognitron, the similarity between learned and input features is measured by the degree of overlap of the two features. Because of the C-cells, it is the blurred versions of the two features that are tested for overlap. Normally features must overlap at their centers. At the peripheries the overlap is minimum. One way to increase the similarity between the two features is to increase the blurring at the peripheries while maintaining the same level of blurring at the center.

Fukushima modifies the normal architecture by incorporating a dual C-cell layer. Suppose layer i of the neocognitron has the dual C-cell layers A and B. C-layer A is going to generate small blurring and C-layer B will generate a large blurring. The receptive field of every S-cell in S-layer $i+1$ will consist of two parts. The center part will get its input from C-layer A (the one having small blurring), while the periphery part will have its input leading from C-layer B. In this way the receptive field will have a small blurring at the center and a large blurring at the peripheries.

After experimenting with the new architecture, Fukushima found that it is only necessary to dualize the first layer. Dualization of other layers did not improve the performance of the neocognitron. Fukushima reports an increase of approximately

7% in performance. In one experiment, the normal neocognitron had an error rate of 16.3% while the dualized neocognitron had an error rate of 9.2% only [FOH94].

Incorporating the dualization idea into our design might help improving the overall performance of the system. This idea sounds appealing because of the large receptive fields that our architecture uses. However some issues have to be resolved before such a modification can take place. The training set has to be modified to suit the new architecture. Along with the training set, the entire set of parameters has to be tuned to fit the new architecture. Our experience tells us that patience should be maintained while performing these two jobs because they consume a very long time. In addition, although the network did not gain any performance increase when layers other than the first one were dualized, dualization can still be incorporated in other layers provided that the blurring mechanism is modified accordingly.

In conclusion, incorporating dualization into our network seems promising. However, many issues have to be resolved before such a goal can be achieved. It is a task to be tested and experimented with in the future.

Appendix A

Necocognitron Training and Running Listing

This program presents the code necessary to train and to run the neocognitron. The preprocessing code is not included. The program gets its input in a character-by-character bases from a text file.

The program has enough switches to control the process of training and running the network

```
/*
        This is the main program. Its objective is to try to recognize the thinned
        characters. The program should be invoked after (thin).

        The program is invoked using the following command:

                run file-0.thin

        where
        file-0.thin:        is the file containing the thinned characters.

        This file may not be specified, in such a case the system will use the default
        name:
                        test-0.thin
        example:
                        run myimage-0.thin,
                or      run (assuming the default name of the file)


        The output to be checked will be stored in file : "result"
        This file will contain the written characters in their corresponding lines.
        Characters that are rejected or confused will be signified by the presence of
        a question mark '?' instead.


        The weights file will be generated in case training is required. In case training
        is not needed this file will be used to load all of the weights.
        Normally, this file is large (around 2.5 MegaBytes). It is, therefore, stored it in the
        (/tmp) directory of the machine being used. It can equally be stored in the normal
        home directory if there is enough space in the account.


        During the operation of the network, the program will display the character that
        it is trying to recognize at the moment.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <strings.h>


#define INPUTFILE "test-0.thin"     /* default input file name      */
#define SUPERFILE "spfinal.1-9"     /* training file                */
#define WFILE    "/tmp/weights"     /* weights after training       */
#define RFILE    "rfinal.1-9"       /* selectivity controllers file */
#define RESLTFILE "result"          /* result file                  */

#define TRAINING 1                  /* 0: no training, 1:do training */

#define ALPHA 0.7                   /* saturation factor            */
#define BASE 0.9                    /* fixed weights initializer    */


#define NUMLAYERS  5                /* number of layers including   */
                                    /* input layer                  */


#define DIMSPLANE0  0               /* dimensions of planes         */
#define DIMCPLANE0  19              /* input plane                  */
```

```
#define DIMSPLANE1   19          /* S-plane of layer 1          */
#define DIMCPLANE1   21          /* C-plane of layer 1          */
#define DIMSPLANE2   19          /* S-plane of layer 2          */
#define DIMCPLANE2   13          /* C-plane of layer 2          */
#define DIMSPLANE3   13          /* S-plane of layer 3          */
#define DIMCPLANE3   7           /* C-plane of layer 3          */
#define DIMSPLANE4   3           /* S-plane of layer 4          */
#define DIMCPLANE4   1           /* C-plane of layer 4          */

#define Q0           0           /* speed of reinforcement      */
#define Q1           1000.0      /* 1st layer                   */
#define Q2           1000.0      /* 2nd layer                   */
#define Q3           1000.0      /* 3rd layer                   */
#define Q4           1000.0      /* 4th layer                   */

#define EXPRMNT 0                /* 0: no experimenting, 1:do expr*/

struct thelimits {               /* The 2D array of variable    */
        int up;                  /* weights might have many     */
        int down;                /* zero elements. These limits */
        int left;                /* enclose all nonzero elements*/
        int right;               /* excluding all possible zero */
};                               /* elements                    */

struct plane {
        float  **pp_cell;        /* pointer to a plane          */
        float  ***ppp_c2s;       /* 1D array of 2D arrays of variable incoming connections*/
        struct thelimits *limits;/* pointer to the limits record*/
        float  v2s;              /* variable inhibitory, from V to S */
        float r;                 /* selectivity controller for this plane */
};
struct slayer {
        int      planewidth,planeheight; /* plane dimensions    */
        int      numplanes;      /* number of planes            */
        struct plane * p_planes; /* an array of planes of this layer */
        float **pp_vc;           /* 2d array of V-cells (inhibotory cells) */
        int      numfields;      /* number of receptive fields  */
        int      fieldsize;      /* dimensions of receptive fields */
        float **pp_c2v;          /* 2D array of fixed incoming connections to the V-cells*/
        float q;                 /* speed of reinforcement      */
        float r;                 /* selectivity controller for the entire layer */
        int step;                /* used in aligning receptive fields */
        int fldstart;            /* used in aligning receptive fields */
};

struct clayer {
        int planewidth,planeheight; /* plane dimensions         */
        int numplanes;           /* number of planes in this layer */
        struct plane * p_planes; /* an array of planes          */
        float **pp_vs;           /* plane of V-cells (inhibitory cells) */
        int numfields;           /* number of receptive fields  */
        int      fieldsize;      /* dimensions of the receptive field */
        float **pp_s2c;          /* 2D array of fixed incoming connections to the C-cell */
        int **pp_join;           /* 2D array of joining of outputs of S-cells */
        int step;                /* used in aligning receptive fields */
        int fldstart;            /* used in aligning receptive fields */
};
struct layer {                   /* layers of the network       */
        struct slayer theslayer; /* S-layer                     */
```

```
        struct clayer theclayer;          /* C-layer                                        */
};

void *reserve_mem(size_t, int );
void initialize_weights(int );                                                              130
float max(float,float);
int   imax(int,int);
 void initialize_weights(int );
 float pos(float );
 void main(int,char **);
 void create_layer(int , struct layer * , int , int , int , int , int , double , double);
 int get_file_pattern(void);
 void generate_outputs(int );
 float outputs(int , int , int , int );
 void update_weights(int , int , int , int );                                               140
 float readc(int , int , int , int );
 float outputc(int , int , int , int );
 float reads(int , int , int , int );
 float outputvc(int ,int ,int );
 float outputvs(int ,int ,int );
 void do_supervised();
 void read_r();
 void save_weights();
 void load_weights();
 void skip_line(FILE * );                                                                   150
 void comp_planes();

struct layer  network[NUMLAYERS];     /* record of network's layers     */

int  NUMSPLANE0;                      /* number of planes               */
int  NUMCPLANE0;                      /* input plane                    */
int  NUMSPLANE1;                      /* of S-layer 1                   */
int  NUMCPLANE1;                      /* of C-layer 1                   */
int  NUMSPLANE2;                      /* of S-layer 2                   */                   160
int  NUMCPLANE2;                      /* of C-layer 2                   */
int  NUMSPLANE3;                      /* of S-layer 3                   */
int  NUMCPLANE3;                      /* of C-layer 3                   */
int  NUMSPLANE4;                      /* of S-layer 4                   */
int  NUMCPLANE4;                      /* of C-layer 4                   */

int FIELDS0    = 0;                   /* number of receptive fields     */
int FIELDC0    = 0;                   /* input plane                    */
int FIELDS1    = 3;                   /* of S-layer 1                   */                   170
int FIELDC1    = 3;                   /* of C-layer 1                   */
int FIELDS2    = 11;                  /* of S-layer 2                   */
int FIELDC2    = 5;                   /* of C-layer 2                   */
int FIELDS3    = 11;                  /* of S-layer 3                   */
int FIELDC3    = 5;                   /* of C-layer 3                   */
int FIELDS4    = 5;                   /* of S-layer 4                   */
int FIELDC4    = 3;                   /* of C-layer 5                   */

long memsize    = 0;                  /* total memory allocated */                          180
long pointermem = 0;                  /* 0    */
long  cellmem   = 0;                  /* 1    */
long weightmem  = 0;                  /* 2    */
long othermem   = 0;                  /* 3    */
FILE *fp;                             /* pointer to input file          */
FILE *fout;                           /* pointer to display file        */
```

```
FILE *fr;                                   /* parameters R file          */
FILE *ftr;                                  /* training patterns file     */
FILE *fres;                                 /* after recognizing the characters, they are written to fres */
                                                                                                          190
int counter1,counter2,counter3,counter4;
int show_pattern=0;
int right,wrong,lamp;
int reject,confuse,substit;
int numlamps;
int totnumlamps;
int passindx;
int anyflag;
int trainplane;

                                                                                                          200
int ***patterns;
float R0,R1,R2,R3,R4;
int numallpatterns,numpatterns;
char inputfile[20];

int COUNTER=20;

int scale[NUMLAYERS];


                                                                                                          210
/*      This is the main function, it calls for initializing the variable weights. Then it calls the
        training function, and finally it runs the network
*/

void main(int argc,char **argv){
int i,j,k,l,m,n;
int num;
int height,width;
char status[10];                                                                                          220
float x=23.34343;
time_t start,finish;
int step;
int active;

fout = stdout;

if(argc == 1)                               /* handles command line       */
        strcpy(inputfile,INPUTFILE);
if(argc == 2)                                                                                             230
        strcpy(inputfile,argv[1]);

fres = fopen(RESULTFILE,"w");

counter1=counter2=counter3=counter4=0;
comp_planes(); /* this is to compute the number of S & C planes in each layer */

/* create all the layers of the network */

create_layer(0,&network[0],NUMSPLANE0,NUMCPLANE0,DIMSPLANE0,DIMCPLANE0,FIELDS0,FIELDC0,R0,Q0);
create_layer(1,&network[1],NUMSPLANE1,NUMCPLANE1,DIMSPLANE1,DIMCPLANE1,FIELDS1,FIELDC1,R1,Q1);
create_layer(2,&network[2],NUMSPLANE2,NUMCPLANE2,DIMSPLANE2,DIMCPLANE2,FIELDS2,FIELDC2,R2,Q2);
create_layer(3,&network[3],NUMSPLANE3,NUMCPLANE3,DIMSPLANE3,DIMCPLANE3,FIELDS3,FIELDC3,R3,Q3);
create_layer(4,&network[4],NUMSPLANE4,NUMCPLANE4,DIMSPLANE4,DIMCPLANE4,FIELDS4,FIELDC4,R4,Q4);

step = 1;
for(i=1;i<NUMLAYERS;i++){                   /* compute the amount of alignments of   */
        step *= network[i].theslayer.step;  /* receptive fields of every layer       */
```

```
        scale[i] = step;
        step *= network[i].theclayer.step;                                    250
}

for(i=1;i<NUMLAYERS;i++)                    /* initialize all the weights      */
        initialize_weights(i);



for(FIELDS1=3 ; FIELDS1>1; FIELDS1 -= 2)    /* in case of analysis these values can be   */
for(FIELDC1=3 ; FIELDC1>1; FIELDC1 -= 2)    /* changed but with care             */260
for(FIELDS2=11; FIELDS2>1; FIELDS2 -= 10)
for(FIELDC2=5 ; FIELDC2>1; FIELDC2 -= 4)
for(FIELDS3=11; FIELDS3>1; FIELDS3 -= 10)
for(FIELDC3=5 ; FIELDC3>1; FIELDC3 -= 4)
for(FIELDS4=5 ; FIELDS4>1; FIELDS4 -= 4)
for(FIELDC4=3 ; FIELDC4>1; FIELDC4 -= 2)
for(R1=0.3;R1>0.1;R1 -= 3.5)
for(R2=1.7;R2>0.1;R2 -= 3.5)
for(R3=1.2;R3>0.1;R3 -= 3.5)
for(R4=1.8;R4>0.1;R4 -= 3.5){                                             270

if(EXPRMNT){
        for(j=0;j<network[1].theslayer.numplanes;j++)
                network[1].theslayer.p_planes[j].r = R1;
        for(j=0;j<network[2].theslayer.numplanes;j++)
                network[2].theslayer.p_planes[j].r = R2;
        for(j=0;j<network[3].theslayer.numplanes;j++)
                network[3].theslayer.p_planes[j].r = R3;
        for(j=0;j<network[4].theslayer.numplanes;j++)
                network[4].theslayer.p_planes[j].r = R4;                      280

}
else
        read_r();

        for(i=1;i<NUMLAYERS;i++)
                initialize_weights(i);

        start = time((time_t *)0);          /* compute training and running time  */
                                                                              290
        if(TRAINING)                        /* When training required (supervised) */
                do_supervised();            /* will be called                  */

        finish= time((time_t *)0);
        fprintf(fout,"it took the training %ld hours, and %ld minutes\n",
                (finish-start)/3600,((finish-start) % 3600)/60);

if(TRAINING){                               /* if training is required, save weights  */
        if(!EXPRMNT)
                save_weights();                                               300
}
else                                        /* otherwise, load weights         */
        load_weights();


/* Run the network */
fprintf(fout,"RUNNING THE NETWORK\n");
show_pattern = 0;
fp = fopen(inputfile,"r");
if(fp == NULL){                                                           310
```

```
                fprintf(fout,"ERROR: input set file\n");
                exit(0);
}
right = wrong = reject = confuse = substit = 0;
totnumlamps=0;
fscanf(fp,"%d ",&numpatterns);
while(get_file_pattern()){                      /* get a character                  */
        generate_outputs(NUMLAYERS-1); /* generate output of all layers */
        if(!EXPRMNT){
                print_clayer(0);                                                        320
/*              print_slayer(1);
                print_clayer(1);
                print_slayer(2);
                print_clayer(2);
                print_slayer(3);
                print_slayer(4);
                print_clayer(4);*/
        }
        for(j=0,numlamps=0;j<network[NUMLAYERS-1].theclayer.numplanes;j++)
                if(network[NUMLAYERS-1].theclayer.p_planes[j].pp_cell[0][0] != 0){      330
                        numlamps++;
                        totnumlamps++;
                        active = j;     /* if numlamps > 1, active = largest active lamp in order */
                }

                                        /* decide if recognition, confusion, rejection,
                                           or substitution                          */
        if(numlamps > 1){               /* if more than on output cell responds     */
                confuse++;              /* then it is a confusion case              */
                strcpy(status,"wrong");                                                 340
        }else
        if(numlamps == 0){              /* if no cell responds, then it is a rejection      */
                reject++;               /* case                                     */
                strcpy(status,"wrong");
        }else
        if(lamp == 0){                  /* a zero was not filtered, so it is a      */
                substit++;              /* substitution case                        */
                strcpy(status,"wrong");
        }else
        if (network[NUMLAYERS-1].theclayer.p_planes[lamp-1].pp_cell[0][0] == 0){        350
                substit++;              /* if the intended cell did not fire, then it       */
                strcpy(status,"wrong"); /* is a substitution case                   */
        }else{
                right++;                /* otherwise, the character has been recognized     */
                strcpy(status,"right"); /* correctly                                */
        }

/*      fprintf(fout,"%d is %s\n",lamp,status);*/

        if(numlamps == 1)                                                               360
                fprintf(fres,"%d ",active+1);
        else
                fprintf(fres,"? ");
}
wrong = reject + confuse + substit;
finish= time((time_t *)0);
fprintf(fout,"it took this run %ld hours, and %ld minutes\n",(finish-start)/3600,((finish-start) % 3600)/60);
start = finish;
fprintf(fout,"**********************************************************************\n\n");
fclose(fp);                                                                             370
fclose(fres);
fflush(fout);
```

```
}

print_c2s(1);
for(i=0;i<2;i++)
        fprintf(fout,"Layer :%d  numplanes = %d.  \n",i,network[i].theclayer.numplanes);
print_v2s();
print_c2v();
print_s2c();                                                                    380

fprintf(fout,"layer1 %d times\nlayer2 %d times\nlayer3 %d times\nlayer4 %d times\n",
                counter1,counter2,counter3,counter4);

finish= time((time_t *)0);
fprintf(fout,"it took the program %ld hours, and %ld minutes\n",(finish-start)/3600,((finish-start) % 3600)/60);
fclose(fout);

exit(2);
                                                                                390
}


/*      This function computes the necessary number of planes of every layer
        The function scans the training file, and from the number of features
        of every layer and the way features are joined, it can decide on the proper
        number of planes of every S and C layer
*/

void comp_planes(){                                                             400
int i,j,k,l,m,n;
int star,joins,height,width,joinc;
char ch;
char line[80],filename[20];
int cplane,splane;
FILE * ftrout;

ftr = fopen(SUPERFILE,"r");
if(ftr == NULL){
        fprintf(fout,"ERROR: training set file\n");                             410
        exit(0);
}

sprintf(filename,"%s%s",SUPERFILE,".out");              /* training file        */
ftrout = fopen(filename,"w");
if(ftrout == NULL){
        fprintf(fout,"ERROR: training set output file error\n");
        exit(0);
}
                                                                                420
NUMSPLANE0 = 0;
NUMCPLANE0 = 1;

for(i=1;i<NUMLAYERS;i++){                       /* loop on all the layers       */

        cplane=0;                               /* which C plane to join to     */
        splane=0;                               /* which S plane to join from   */

        while(1){
                                                                                430
                if(fgets(line,80,ftr) == NULL){
                        fprintf(fout,"ERROR: training set incomplete\n");
                        exit(0);
                }
```

```
                if(!strncmp(line,"*",1)){
                        if(sscanf(line,"%c %d %d %d %d ",&star,&joins,&height,&width,&joinc) == EOF){
                                fprintf(fout,"ERROR: training set incomplete\n");
                                exit(0);
                        }                                                                          440
                        fprintf(ftrout,"%c %d %d %d %d \n",star,splane,height,width,cplane);

                           if(!joins)
                                splane++;
                           if(!joinc)
                                cplane++;
                }
                else{
                        for(j=0;j<width;j++)
                                switch(line[j]){                                                   450
                                        case '0': line[j] = '.';break;
                                        case '1': line[j] = '*';break;
                                        case 'x': line[j] = '.';break;
                                        case 'X': line[j] = '*';break;
                                };
                        fputs(line,ftrout);
                }
                if(!strncmp(line,"%%",2))  /* training set of layer i finished */
                        break;
                                                                                                   460
        }
        switch(i){
        case 1: NUMSPLANE1 = splane; NUMCPLANE1 = cplane; break;
        case 2: NUMSPLANE2 = splane; NUMCPLANE2 = cplane; break;
        case 3: NUMSPLANE3 = splane; NUMCPLANE3 = cplane; break;
        case 4: NUMSPLANE4 = splane; NUMCPLANE4 = cplane; break;
        }
}
fclose(ftr);
fclose(ftrout);                                                                                    470
}

/*
        This function reads values of the selectivity controllers (r) from
        the theri file
*/

void read_r(){
int i,j,k;
int plane;                                                                                         480
char star;
float rplane,rlayer;
char line[80];


if((fr = fopen(RFILE,"r")) == NULL){
        printf("ERROR : r file\n");
        exit(0);
}
                                                                                                   490
for(i=1;i<NUMLAYERS;i++)                        /* loop on all layers       */
        while(1) {

                if(fgets(line,80,fr) == NULL){
                        fprintf(fout,"ERROR: r file incomplete\n");
                        exit(0);
```

```
                }
                if(!strncmp(line,"%%",2))  /* training set of layer i finished */
                        break;
                if(!strncmp(line,"0",1))   /* a comment */                          500
                        ;
                else
                if(!strncmp(line,"*",1)) {
                        if(sscanf(line,"%c %f ",&star,&rlayer) == EOF){
                                fprintf(fout,"ERROR: training set incomplete\n");
                                exit(0);
                        }
                        for(j=0;j<network[i].theslayer.numplanes;j++)
                                network[i].theslayer.p_planes[j].r = rlayer;
                }                                                                 510
                else {
                        if(sscanf(line,"%d %f ",&plane,&rplane) == EOF){
                                fprintf(fout,"ERROR : r file error\n");
                                exit(0);
                        }
                        network[i].theslayer.p_planes[plane].r = rplane;
                }

        }                                                                         520
fclose(fr);

}

/*
        In case no training is required, the network can use the saved weights file
        the file is very large (about 2.5 Mega Bytes), so usually we save it in the
        /tmp directory of any Unix machine

*/                                                                                530

void load_weights(){
int i,j,k,m,n;
int field,up,down,left,right;
float val;
int ival;
FILE *ftr;
char line[200];

/* S2C */                                                                         540

if((ftr = fopen(WFILE,"r")) == NULL){
        fprintf(fout,"ERROR: Weights File Error\n");
        exit(0);
}
skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){               /* the S2C fixed weights excitatory      */
        skip_line(ftr);
        for(j=0;j<network[i].theclayer.fieldsize;j++)
                for(k=0;k<network[i].theclayer.fieldsize;k++){                     550
                        fscanf(ftr,"%f",&val);
                        network[i].theclayer.pp_s2c[j][k] = val;
                }
        skip_line(ftr); /* remove new line char */

}

/* C2V */
```

```
skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){                    /* the C2V fixed weights                    */
        skip_line(ftr);
        for(j=0;j<network[i].theslayer.fieldsize;j++)
                for(k=0;k<network[i].theslayer.fieldsize;k++){
                        fscanf(ftr,"%f",&val);
                        network[i].theslayer.pp_c2v[j][k] = val;
                }
        skip_line(ftr); /* remove new line char */

}

/* V2S */
skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){                    /* the V2S variable inhibitory weights      */
        skip_line(ftr);
        for(j=0;j<network[i].theslayer.numplanes;j++){
                fscanf(ftr,"%f",&val);
                network[i].theslayer.p_planes[j].v2s = val;
        }
        skip_line(ftr); /* remove new line char */
}


/* C2S */

skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){                    /* the C2S variable excitatory weights      */
        skip_line(ftr);
        for(j=0;j<network[i].theslayer.numplanes;j++){
                skip_line(ftr);
                for(k=0;k<network[i].theslayer.numfields;k++){
                        skip_line(ftr);
                        for(m=0;m<network[i].theslayer.fieldsize;m++)
                                for(n=0;n<network[i].theslayer.fieldsize;n++){
                                        fscanf(ftr,"%f",&val);
                                        network[i].theslayer.p_planes[j].ppp_c2s[k][m][n] = val;
                                }
                        skip_line(ftr); /* remove new line char */

                }
        }
}

/* Limits */
skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){                    /* limits of receptive fields               */
        skip_line(ftr);
        for(j=0;j<network[i].theslayer.numplanes;j++){
                skip_line(ftr);
                for(k=0;k<network[i].theslayer.numfields;k++){
                        fscanf(ftr,"Field %2d, Up = %2d, Down = %2d, Left = %2d, Right = %2d ",
                                        &field,&up,&down,&left,&right);
                        network[i].theslayer.p_planes[j].limits[k].up = up;
                        network[i].theslayer.p_planes[j].limits[k].down = down;
                        network[i].theslayer.p_planes[j].limits[k].left = left;
                        network[i].theslayer.p_planes[j].limits[k].right = right;
                }
        }
}
```

```
skip_line(ftr);
for(i=1;i<NUMLAYERS;i++){                /* the joining array                */
        for(j=0;j<network[i].theslayer.numplanes;j++)
                for(k=0;k<network[i].theclayer.numplanes;k++){
                        fscanf(ftr,"%d ",&ival);
                        network[i].theclayer.pp_join[j][k] = ival;
                }                                                              630
}


}

/*
        This function skips onle line in the weights file. This is to skip
        text lines
*/
void skip_line(FILE * fptr){                                                   640
char line[200];
        if(fgets(line,200,fptr) == NULL){
                printf("ERROR: File error\n");
                exit(0);
        }
}


/*
        This function is usually called after training is finished. Upon calling it,   650
        all the weights will be saved, preferably in the /tmp directory of any
        Unix machine.
*/
void save_weights(){
int i,j,k,m,n;
int up,down,left,right;
FILE *ftr;

ftr = fopen(WFILE,"w");
                                                                               660
fprintf(ftr,"S2C weights (fixed)\n");
for(i=1;i<NUMLAYERS;i++){                /* the S2C fixed excitatory weights    */
        fprintf(ftr,"Layer %d\n",i);
        for(j=0;j<network[i].theclayer.fieldsize;j++){
                for(k=0;k<network[i].theclayer.fieldsize;k++)
                        fprintf(ftr,"%6.5f ",network[i].theclayer.pp_s2c[j][k]);
                fprintf(ftr,"\n");
        }
}
                                                                               670

fprintf(ftr,"C2V weights (fixed)\n");
for(i=1;i<NUMLAYERS;i++){                /* the C2V fixed weights               */
        fprintf(ftr,"Layer %d \n",i);
        for (j=0;j<network[i].theslayer.fieldsize;j++){
                for(k=0;k<network[i].theslayer.fieldsize;k++)
                        fprintf(ftr,"%6.5f ",network[i].theslayer.pp_c2v[j][k]);
                fprintf(ftr,"\n");
        }
                                                                               680
}
```

```
fprintf(ftr,"V2S weights (variable)\n");
for(i=1;i<NUMLAYERS;i++){              /* the V2S inhibitory weights              */
        fprintf(ftr,"Layer %d\n",i);
        for(j=0;j<network[i].theslayer.numplanes;j++)
                fprintf(ftr,"%14.5f\n",network[i].theslayer.p_planes[j].v2s);

}
                                                                                    690

fprintf(ftr,"C2S weights (variable)\n");
for(i=1;i<NUMLAYERS;i++){              /* the C2S variable excitatory weights     */
        fprintf(ftr,"Layer %d\n",i);
        for(j=0;j<network[i].theslayer.numplanes;j++){
                fprintf(ftr,"Plane %d\n",j);
                for(k=0;k<network[i].theslayer.numfields;k++){
                        fprintf(ftr,"Field %d\n",k);
                        for(m=0;m<network[i].theslayer.fieldsize;m++){
                                for(n=0;n<network[i].theslayer.fieldsize;n++)                    700
                                        fprintf(ftr,"%11.5f ",network[i].theslayer.p_planes[j].ppp_c2s[k][m][n]);
                                fprintf(ftr,"\n");
                        }
                }
        }
}

fprintf(ftr,"Limits\n");
for(i=1;i<NUMLAYERS;i++){              /* limits of receptive fields              */
        fprintf(ftr,"Layer %d\n",i);                                                710
        for(j=0;j<network[i].theslayer.numplanes;j++){
                fprintf(ftr,"Plane %d\n",j);
                for(k=0;k<network[i].theslayer.numfields;k++){
                        up      = network[i].theslayer.p_planes[j].limits[k].up;
                        down    = network[i].theslayer.p_planes[j].limits[k].down;
                        left    = network[i].theslayer.p_planes[j].limits[k].left;
                        right   = network[i].theslayer.p_planes[j].limits[k].right;
                        fprintf(ftr,"Field %2d, Up = %2d, Down = %2d, Left = %2d, Right = %2d\n",
                                        k,up,down,left,right);
                }                                                                   720
        }
}


fprintf(ftr,"join array\n");
for(i=1;i<NUMLAYERS;i++){              /* joining array                           */
        for(j=0;j<network[i].theslayer.numplanes;j++)  {
                for(k=0;k<network[i].theclayer.numplanes;k++)
                        fprintf(ftr,"%d ",network[i].theclayer.pp_join[j][k]);
                fprintf(ftr,"\n");                                                  730
        }
}

fclose(ftr);

}


/*
        This is the supervised training function. Its job is to train all layers of the     740
        network. After it finishes, the C2S and the V2S weights will have their values
        modified according to the training features.
*/
```

```
void do_supervised(){
int i,j,k,l,m,n;
int star,joins,height,width,joinc;
char ch;
char line[80];
int cplane,splane;                                                    750
int reprow,repcol;
int rowDist,colDist;
int neg;
int up,down,left,right;
int plane,field;
struct slayer * p_slayer;
struct clayer * p_clayer;

ftr = fopen(SUPERFILE,"r");
if(ftr == NULL){                                                      760
        fprintf(fout,"ERROR: training set file\n");
        exit(0);
}

for(i=1;i<NUMLAYERS;i++){           /* loop on all the layers        */

        p_slayer = &(network[i].theslayer);
        p_clayer = &(network[i].theclayer);

                                                                      770
        for(m=0;m<DIMCPLANE0;m++) /* empty the planes                */
                for(n=0;n<DIMCPLANE0;n++)
                        network[0].theclayer.p_planes[0].pp_cell[m][n] = 0;

        cplane=0; /* which C plane to join to */
        splane=0; /* which S plane to join from */


        while(1){
                                                                      780
                reprow = network[i].theslayer.planeheight/2; /* coordinates of representative*/
                repcol = network[i].theslayer.planewidth/2;  /* cell                          */

                if(fgets(line,80,ftr) == NULL){
                        fprintf(fout,"ERROR: training set incomplete\n");
                        exit(0);
                }
                if(!strncmp(line,"%%",2))  /* training set of layer i finished */
                        break;
                if(strncmp(line,"*",1)){    /* beginning of a training pattern */   790
                        fprintf(fout,"ERROR: beginning of training pattern expected\n");
                        exit(0);
                }
                if(sscanf(line,"%c %d %d %d %d ",&star,&joins,&height,&width,&joinc) == EOF){
                        fprintf(fout,"ERROR: training set incomplete\n");
                        exit(0);
                }

                /* read the training patterns       */
                                                                      800
                for(m=(DIMCPLANE0-height)/2;m<(DIMCPLANE0+height)/2;m++)
                        for(n=(DIMCPLANE0-width)/2;n<(DIMCPLANE0+width)/2;n++){

                                if(fscanf(ftr,"%c ",&ch) == EOF){
                                        fprintf(fout,"ERROR: training set incomplete\n");
                                        exit(0);
```

```
                                }
                                if((ch == 'x') || (ch == 'X')){
                                        rowDist = min((abs(m-DIMCPLANE0/2))/scale[i],
                                                        network[i].theslayer.planeheight/2);
                                        if(m<DIMCPLANE0/2) rowDist *= -1;
                                        colDist = min((abs(n-DIMCPLANE0/2))/scale[i],
                                                        network[i].theslayer.planewidth/2);
                                        if(n<DIMCPLANE0/2) colDist *= -1;
                                        reprow += rowDist;
                                        repcol += colDist;
                                        if(ch == 'x')
                                                network[0].theclayer.p_planes[0].pp_cell[m][n] = 0;
                                        else
                                                network[0].theclayer.p_planes[0].pp_cell[m][n] = 1;
                                }
                                else
                                        network[0].theclayer.p_planes[0].pp_cell[m][n] = ch-'0';


                        }

/*          print_clayer(0);*/
            generate_outputs(i-1);          /* generate outputs of all previous layers          */

            update_weights(i,splane,reprow,repcol); /* update weights of current layer          */

            network[i].theclayer.pp_join[splane][cplane] = 1;

            if(!joins)
                        splane++;
            if(!joinc)
                        cplane++;
        }

        for(plane=0;plane<p_slayer->numplanes;plane++)  /* after updating the weights, compute */
                for(field=0;field<p_slayer->numfields;field++){ /* limits of the receptive fields */
                        up = left = p_slayer->fieldsize-1;
                        down = right = 0;
                        for(j=0;j<p_slayer->fieldsize;j++)
                                for(k=0;k<p_slayer->fieldsize;k++)
                                        if(p_slayer->p_planes[plane].ppp_c2s[field][j][k] != 0){
                                                if(j < up)
                                                        up = j;
                                                if(k < left)
                                                        left = k;
                                                if(j > down)
                                                        down = j;
                                                if(k > right)
                                                        right = k;
                                        }
                        p_slayer->p_planes[plane].limits[field].up    = up;
                        p_slayer->p_planes[plane].limits[field].down  = down;
                        p_slayer->p_planes[plane].limits[field].left  = left;
                        p_slayer->p_planes[plane].limits[field].right = right;

                }


if(!EXPRMNT)
        for(m=0;m<network[i].theslayer.numplanes;m++)  {
                for(n=0;n<network[i].theclayer.numplanes;n++)
                        printf("%d",network[i].theclayer.pp_join[m][n]);
```

```
                printf("\n");
        }
}
fclose(ftr);
}



/*
        This function gets input characters from input file
*/


int get_file_pattern(void){

int i,j,k;
int height,width;
char star,ch;
char line[100];


if(fgets(line,100,fp) == NULL)
        return(0);

while(!strncmp(line,"END OF LINE",11) || !strncmp(line,"CHAR 0",6)){

        if(!strncmp(line,"END OF LINE",11))
                fprintf(fres,"\n\n");
        if(!strncmp(line,"CHAR 0",6))
                fprintf(fres,"0 ");

        if(fgets(line,100,fp) == NULL)
                return(0);
}


height = width = 0;
if(sscanf(line,"%c %d %d %d ",&star,&height,&width,&lamp) == EOF)
        return(0);

if(show_pattern)
        fprintf(fout,"Star: %c,   height: %d,   width: %d\n",star,height,width);


while((height != DIMCPLANE0) || (width != DIMCPLANE0)){
        fprintf(fout,"ERROR: Incompatability with input plane sizes\n");
        if(fscanf(fp,"%c %d %d %d ",&star,&height,&width,&lamp) == EOF)
                return(0);
        if(show_pattern)
                fprintf(fout,"Star: %c,   height: %d,   width: %d\n",star,height,width);

}

if(show_pattern)
        for(i=0;i<DIMCPLANE0;i++){
                for(j=0;j<DIMCPLANE0;j++){
                        if(fscanf(fp,"%c ",&ch) == EOF){
                                fprintf(fout,"ERROR: Incomplete Input \n");
                                break;
```

```
                        }
                        if(!(ch-'0'))
                                fprintf(fout,".");
                        else
                                fprintf(fout,"*");

                        network[0].theclayer.p_planes[0].pp_cell[i][j] =ch-'0';
                }
                fprintf(fout,"\n");
        }
else
        for(i=0;i<DIMCPLANE0;i++){
                for(j=0;j<DIMCPLANE0;j++){
                        if(fscanf(fp,"%c ",&ch) == EOF){
                                fprintf(fout,"ERROR: Incomplete Input \n");
                                break;
                        }
                        network[0].theclayer.p_planes[0].pp_cell[i][j] =ch-'0';
                }
        }

if(show_pattern)
        fprintf(fout,"-------------------------------------\n");


return(1);
}


/*
        This function will update the variable weights
        of the corresponding representative of one plane of one layer
*/
void update_weights(int layer,int plane,int row,int column){
int i,j,k;
int strow,stcol;
struct slayer * p_slayer;
struct clayer * p_clayer;
int up,down,left,right;

switch(layer){
        case 1:counter1++;break;
        case 2:counter2++;break;
        case 3:counter3++;break;
        case 4:counter4++;break;
}
p_slayer = &(network[layer].theslayer);
p_clayer = &(network[layer].theclayer);

strow = row    * p_slayer->step + p_slayer->fldstart;
stcol = column * p_slayer->step + p_slayer->fldstart;
for(i=0;i<p_slayer->numfields;i++)
        for(j=0;j<p_slayer->fieldsize;j++)
                for(k=0;k<p_slayer->fieldsize;k++)
                        p_slayer->p_planes[plane].ppp_c2s[i][j][k] +=
                                p_slayer->q * p_slayer->pp_c2v[j][k] * readc(layer-1,i,strow+j,stcol+k);

p_slayer->p_planes[plane].v2s += p_slayer->q * p_slayer->pp_vc[row][column];


}
```

```
/*
        Upon projecting one training pattern (wether local or global) on the input
        plane for training layer i, this function will generate outputs of all layers
        preceding layer i

        This function is also used when running the network. When projecting one character
        on the input plane, this function will be called to produce the reposne of the
        entire network                                                          1000
*/
void generate_outputs(int layer){
/* layer 0 is not processed because it's the input layer */
int i,j,k;
int row,col;
int strwo,stcol;
float sum1,sum2,sum3;
struct slayer * p_slayer;
struct clayer * p_clayer;
                                                                                1010

for(i=1;i<=layer;i++) {
        p_slayer = &(network[i].theslayer);
        p_clayer = &(network[i].theclayer);

    for(row=0;row<p_slayer->planeheight;row++)
        for(col=0;col<p_slayer->planewidth;col++)
            p_slayer->pp_vc[row][col] = outputvc(i,row,col);

    for(j=0;j<p_slayer->numplanes;j++)                                          1020
        for(row=0;row<p_slayer->planeheight; row++)
            for(col=0;col<p_slayer->planewidth;col++)
                p_slayer->p_planes[j].pp_cell[row][col] = outputs(i,j,row,col);

    for(row=0;row<p_clayer->plancheight;row++)
        for(col=0;col<p_clayer->planewidth;col++)
            if(i == NUMLAYERS-1)    /* only for last layer */
                p_clayer->pp_vs[row][col] = outputvs(i,row,col);
            else
                p_clayer->pp_vs[row][col] = 0;                                  1030

    for(j=0;j<p_clayer->numplanes;j++)
        for(row=0;row<p_clayer->planeheight; row++)
            for(col=0;col<p_clayer->planewidth;col++)
                p_clayer->p_planes[j].pp_cell[row][col] = outputc(i,j,row,col);


}

if(layer != NUMLAYERS-1){                                                       1040
        p_slayer = &(network[layer+1].theslayer);
        for(row=0;row<p_slayer->planeheight;row++)
                for(col=0;col<p_slayer->planewidth;col++)
                        p_slayer->pp_vc[row][col] = outputvc(layer+1,row,col);
}

}


/*                                                                              1050
        This function computes the output of the V-cell (inhibitory Cell) to be used
        in computing the output of the corresponding S-cell
*/
float outputvc(int layer,int row,int column){
```

```
int i,j,k;
float sum1;
float coutput;
struct slayer * p_slayer;
struct clayer * p_clayer;
int strow,stcol;                                                    1060

p_slayer = &(network[layer].theslayer);
p_clayer = &(network[layer].theclayer);

sum1 = 0;

strow = row    * p_slayer->step + p_slayer->fldstart;
stcol = column * p_slayer->step + p_slayer->fldstart;

for(i=0;i<p_slayer->numfields;i++)                                  1070
        for(j=0;j<p_slayer->fieldsize;j++)
                for(k=0;k<p_slayer->fieldsize;k++){

            coutput = readc(layer-1,i,strow+j,stcol+k);
            sum1 += p_slayer->pp_c2v[j][k] * coutput * coutput;

                }

return((float)sqrt(sum1));
}                                                                   1080

/*
        This function computes the output of laterla inhibition. Lateral inhibition is only
        used in the recognition layer
*/
float outputvs(int layer,int row,int column){
int i,j,k;
float sum,sum1,sum2;
float r,tmp;
struct slayer * p_slayer;                                           1090
struct clayer * p_clayer;
int strow,stcol;

p_slayer = &(network[layer].theslayer);
p_clayer = &(network[layer].theclayer);

strow = row    * p_clayer->step + p_clayer->fldstart;
stcol = column * p_clayer->step + p_clayer->fldstart;
sum2=0;
                                                                    1100
for(i=0;i<p_slayer->numplanes;i++)
        for(j=0;j<p_clayer->fieldsize;j++)
                for(k=0;k<p_clayer->fieldsize;k++)
                        sum2 += p_clayer->pp_s2c[j][k] * reads(layer,i,strow+j,stcol+k);

if(sum2 != 0){
        sum2 = sum2 / p_clayer->numplanes;
}
return(sum2);
                                                                    1110
}


/*
        This function computes the output of C-cells. It uses the fixed excitatory weight
        leading from the S-cells in the preceding S-sublayer
```

```
*/

float outputc(int layer,int plane,int row,int column){
int i,j,k,s_index;                                                    1120
float sum,sum1,sum2;
float r,tmp;
struct slayer * p_slayer;
struct clayer * p_clayer;
int strow,stcol;

p_slayer = &(network[layer].theslayer);
p_clayer = &(network[layer].theclayer);

strow = row    * p_clayer->step + p_clayer->fldstart;                 1130
stcol = column * p_clayer->step + p_clayer->fldstart;

sum1=0; /* loop on all s planes */

for(s_index=0;s_index<p_slayer->numplanes;s_index++)
        if(p_clayer->pp_join[s_index][plane])
                for(i=0;i<p_clayer->fieldsize;i++)
                        for(j=0;j<p_clayer->fieldsize;j++)
                                sum1 += p_clayer->pp_s2c[i][j] * reads(layer,s_index,strow+i,stcol+j);
                                                                      1140

tmp = max((1 + sum1)/(1 + p_clayer->pp_vs[row][column]) - 1,0);
tmp = (tmp/(ALPHA + tmp));
return(tmp);

}

/*
        This function reads the the contents of a particular S-cell
*/                                                                    1150
float reads(int layer,int plane,int row,int column){

if(     (row < network[layer].theslayer.planeheight) &&
        (row >= 0) &&
        (column < network[layer].theslayer.planewidth) &&
        (column >= 0))
                return(network[layer].theslayer.p_planes[plane].pp_cell[row][column]);
else
                return(0);
}                                                                     1160


/*
        This function computes the output of a particular S-cell. It makes use
        of the variable excitatory and inhibitory weights
*/

float outputs(int layer,int plane,int row,int column){
int i,j,k;
int anyone=0;                                                         1170
float sum1,sum2,sum3;
float r;
float tmp1,tmp2,tmp3,tmp4,tmpres,coutput;
struct slayer * p_slayer;
struct clayer * p_clayer;
int strow,stcol;

p_slayer = &(network[layer].theslayer);
```

```
p_clayer = &(network[layer].theclayer);
                                                                        1180
r =p_slayer->p_planes[plane].r;
sum1 = sum2 = 0;


strow = row    * p_slayer->step + p_slayer->fldstart;
stcol = column * p_slayer->step + p_slayer->fldstart;

for(i=0;i<p_slayer->numfields;i++)
        for(j=p_slayer->p_planes[plane].limits[i].up; j<=p_slayer->p_planes[plane].limits[i].down; j++)
                for(k=p_slayer->p_planes[plane].limits[i].left; k<=p_slayer->p_planes[plane].limits[i].right;k++){
                        coutput = readc(layer-1,i,strow+j,stcol+k);
                        sum1 += p_slayer->p_planes[plane].ppp_c2s[i][j][k] * coutput;
                }


tmp1 = 1 + sum1;
tmp2 = 1 + (r/(r+1)) * p_slayer->p_planes[plane].v2s * p_slayer->pp_vc[row][column];
tmpres = r*max(tmp1/tmp2 - 1,0);

                                                                        1200
return(tmpres);



}

/*
        This function gets the value of one C-cell from the specified plane
*/
float readc(int layer,int plane,int row,int column){                    1210

if(     (row < network[layer].theclayer.planeheight) &&
        (row >= 0) &&
        (column < network[layer].theclayer.planewidth) &&
        (column >= 0))
                return(network[layer].theclayer.p_planes[plane].pp_cell[row][column]);
else
                return(0);
}
                                                                        1220

float max(float x,float y){
if (x >= y)
        return(x);
else
        return(y);
}

int imax(int x,int y){
if (x >= y)                                                             1230
        return(x);
else
        return(y);
}

/*
        This function creates layers of the network
*/
void create_layer(layernum,p_layer,numsplane,numcplane,dimsplane,dimcplane,sfield,cfield,r,q)
int layernum;                                                           1240
```

```
struct layer *p_layer;
int numsplane,numcplane,dimsplane,dimcplane;
int sfield,cfield;
double r,q;
{
   int i,j,k,l;
   float t1,t2;
   int t3;
   struct slayer * p_slayer;
   struct clayer * p_clayer;                                                    1250

        p_slayer = &p_layer->theslayer;
        p_clayer = &p_layer->theclayer;

        p_slayer->planewidth     = dimsplane;
        p_slayer->planeheight    = dimsplane;
        p_slayer->numplanes      = numsplane;
        p_slayer->numfields      = 0;
        p_slayer->fieldsize      = sfield;
        p_slayer->r                     = (float)r;                             1260
        p_slayer->q                     = (float)q;
        if(layernum != 0) {

                t1 = network[layernum-1].theclayer.planeheight - p_slayer->fieldsize;
                t2 = dimsplane - 1;
                if(t2==0)
                        t3 = 1;  /* the value is immaterial */
                else
                        t3 = (int)ceil(t1/t2);
                p_slayer->step = min(t3,sfield);                                1270
                p_slayer->step = imax(1,p_slayer->step);

                t1 = (float)(network[layernum-1].theclayer.planeheight / 2);
                t1 += (float)(sfield/2) + 1;
                t2 = (int)((dimsplane+1)/2) * p_slayer->step + sfield - p_slayer->step;
                p_slayer->fldstart = (int)(t1 - t2);


                p_slayer->numfields = network[layernum-1].theclayer.numplanes;
                                                                                1280
                p_slayer->pp_c2v = (float **)reserve_mem(sfield*sizeof(float *),0);
                for(i=0;i<sfield;i++)
                        p_slayer->pp_c2v[i]=(float *)reserve_mem(sfield*sizeof(float),2);

        p_slayer->pp_vc = (float **)reserve_mem(dimsplane * sizeof(float *),0);
        for(i=0;i<dimsplane;i++)
            p_slayer->pp_vc[i] = (float *)reserve_mem(dimsplane*sizeof(float),1);

                p_slayer->p_planes = (struct plane *)reserve_mem(numsplane*sizeof(struct plane),3);
                for (i=0;i<numsplane;i++) {                                     1290
                        p_slayer->p_planes[i].pp_cell = (float **)reserve_mem(dimsplane * sizeof(float *),0);
                        for(j=0;j<dimsplane;j++){

                p_slayer->p_planes[i].pp_cell[j] =(float *)reserve_mem(dimsplane*sizeof(float),1);

                                for(k=0;k<dimsplane;k++)
                                        p_slayer->p_planes[i].pp_cell[j][k] = j+k;
                        }

                        p_slayer->p_planes[i].ppp_c2s =                         1300
                                (float ***)reserve_mem(p_slayer->numfields * sizeof(float **),0);
                        for(j=0;j<p_slayer->numfields;j++){
```

```
                        p_slayer->p_planes[i].ppp_c2s[j] =
                                (float **)reserve_mem(sfield * sizeof(float *),0);
                        for(k=0;k<sfield;k++){
                                p_slayer->p_planes[i].ppp_c2s[j][k] =
                                        (float *)reserve_mem(sfield * sizeof(float),2);
                                for(l=0;l<sfield;l++)
                                        p_slayer->p_planes[i].ppp_c2s[j][k][l] = 0;
                        }                                                                    1310
                }
                p_slayer->p_planes[i].limits = (struct thelimits *)
                        reserve_mem(p_slayer->numfields * sizeof(struct thelimits),1);

        }
    }



                                                                                             1320
    p_clayer->planewidth = dimcplane;
    p_clayer->planeheight = dimcplane;
    p_clayer->numplanes = numcplane;

    if(layernum != 0) {
            p_clayer->numfields = 1;
            p_clayer->pp_s2c = (float **)reserve_mem(cfield*sizeof(float *),0);
            for(i=0;i<cfield;i++)
                    p_clayer->pp_s2c[i]=(float *)reserve_mem(cfield*sizeof(float),2);         1330
    }
    else
            p_clayer->numfields = 0;

    t1 = dimsplane - cfield;
    t2 = dimcplane - 1;
    if(t2 == 0)
            t3 = 1; /* the value is immaterial */
    else
            t3 = (int)ceil(t1/t2);                                                           1340
    p_clayer->step = min(t3,cfield);
    p_clayer->step = imax(1,p_clayer->step);


    t1 = (float)(dimsplane / 2);
    t1 += (float)(cfield/2) + 1;
    t2 = (int)((dimcplane+1)/2) * p_clayer->step + cfield - p_clayer->step;
    p_clayer->fldstart = (int)(t1 - t2);

                                                                                             1350
    p_clayer->fieldsize         =       cfield;

p_clayer->pp_vs = (float **)reserve_mem(dimcplane * sizeof(float *),0);
for(i=0;i<dimcplane;i++)
    p_clayer->pp_vs[i] = (float *)reserve_mem(dimcplane*sizeof(float),1);

p_clayer->p_planes = (struct plane *)reserve_mem(numcplane*sizeof(struct plane),3);
    for (i=0;i<numcplane;i++) {
            p_clayer->p_planes[i].pp_cell = (float **)reserve_mem(dimcplane * sizeof(float *),0);   1360
            for(j=0;j<dimcplane;j++){
                    p_clayer->p_planes[i].pp_cell[j] =(float *)reserve_mem(dimcplane*sizeof(float),1);
                    for(k=0;k<dimcplane;k++)
                            p_clayer->p_planes[i].pp_cell[j][k] = j+k;
```

```
                }

        }

        if(layernum != 0){                                                      1370
                p_clayer->pp_join = (int **)reserve_mem(numsplane * sizeof(int *),0);
                for(i=0;i<numsplane;i++)
                        p_clayer->pp_join[i] = (int *)reserve_mem(numcplane * sizeof(int),1);
                for(i=0;i<numsplane;i++)
                        for(j=0;j<numcplane;j++)
                                p_clayer->pp_join[i][j] = 0;
        }

        fprintf(fout,"..%ld \n",(long)(memsize/sizeof(float)));
                                                                                1380
}


int min(int x,int y){

        if(x < y)
                return(x);
        else
                return(y);
                                                                                1390
}

/*
        this function initializes fixed and variable weights. Variable weights are initialized
        to zero
*/
void initialize_weights(int layernum){

   int i,j,k,l;
   float weight;                                                                1400
   struct slayer * p_slayer;
   struct clayer * p_clayer;
   int sfcntr,cfcntr;                   /* center of the field */
   FILE *fweights;
   int weights[10][10],val;
   float  weightsf[10][10],valf;
   float tempsum;



                                                                                1410
        p_slayer = &network[layernum].theslayer;
        p_clayer = &network[layernum].theclayer;
        sfcntr = (p_slayer->fieldsize)/2;
        cfcntr = (p_clayer->fieldsize)/2;


        /* initialize fixed connections */

        if(BASE == 1) {
                weight = (float)1 / (p_slayer->numfields * p_slayer->fieldsize * p_slayer->fieldsize);  1420
                for(i=0;i<p_slayer->fieldsize;i++)
                        for(j=0;j<p_slayer->fieldsize;j++)
                                p_slayer->pp_c2v[i][j] = weight;
        }
        else {
                tempsum = 0;
```

```
                for(i=0;i<p_slayer->fieldsize;i++)
                        for(j=0;j<p_slayer->fieldsize;j++) {
                                p_slayer->pp_c2v[i][j] = (float)pow((double)BASE,
                                        (double)sqrt((double)(i-sfcntr)*(i-sfcntr)+(j-sfcntr)*(j-sfcntr)));
                                tempsum += p_slayer->pp_c2v[i][j];
                        }
        }

        if(BASE == 1) {
                weight = (float)1/(p_clayer->fieldsize * p_clayer->fieldsize);
                for(i=0;i<p_clayer->fieldsize;i++)
                        for(j=0;j<p_clayer->fieldsize;j++)
                                p_clayer->pp_s2c[i][j] = weight;                                        1440
        }
        else {
                tempsum = 0;
                for(i=0;i<p_clayer->fieldsize;i++)
                        for(j=0;j<p_clayer->fieldsize;j++){
                                p_clayer->pp_s2c[i][j] = (float)pow((double)BASE,
                                        (double)sqrt((double)(i-cfcntr)*(i-cfcntr)+(j-cfcntr)*(j-cfcntr)));
                                tempsum += p_clayer->pp_s2c[i][j];
                        }
                                                                                                        1450
        }



        /* initialize variable connections  */

        for(i=0;i<p_slayer->numplanes;i++)
                for(j=0;j<p_slayer->numfields;j++)
                        for(k=0;k<p_slayer->fieldsize;k++)                                              1460
                                for(l=0;l<p_slayer->fieldsize;l++)
                                        p_slayer->p_planes[i].ppp_c2s[j][k][l] = 0;


        for(i=0;i<p_slayer->numplanes;i++)
                p_slayer->p_planes[i].v2s = 0;

}

void *reserve_mem(size_t size,int memkind)                                                              1470
{
        void *s;
        s=malloc(size);
        if (s==NULL)
        {
                fprintf(fout,"\nFATAL ERROR: OUT OF MEMORY\n");
                        exit(10);
        }
        switch(memkind){
                case 0 : pointermem += size;break;                                                      1480
                case 1:  cellmem += size;break;
                case 2: weightmem += size;break;
                case 3: othermem += size;break;
        }
        memsize += size;
        return s;
}
```

Bibliography

[AKHM80] Adnan Amin, Azzedine Kaced, Jean-Paul Haton, and Roger Mohr. Hand written arabic character recognition by the IRAC system. In *Proceedings of the 5th Internationl Joint Conference on Pattern Recognition*, pages 729–731, December 1980.

[AMSH89] Hazem Y. Abdelazim, A. M. Mousa, Y. R. Saleh, and M. A. Hashish. Arabic text recognition using a partial observation approach. In *Proceedings of the 12th National Computer Conference*, pages 427–437, 1989.

[AYU92] H. Al-Yousefi and S. S. Upda. Recognition of arabic characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(8):853–857, August 1992.

[BD79] Evan L. Brown and Kenneth Deffenbacher. *Perception and the Senses.* Oxford University Press, 1979.

[Bok92] Mindy Bokser. Omnidocument technologies. *Proceedings of the IEEE*, 80(7):1066–1078, July 1992.

[Bow92] Sing-Tze Bow. *Pattern Recognition and Image Preprocessing.* Marcel Dekker, Inc., 270 Madison Avenue, New York, 1992.

[Bra93] John Bradley. Zip code breakers. *Scientific American*, pages 102–103, February 1993.

[CBD+89] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and H. S. Baird. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, I:541–551, 1989.

[CL94] Ju Wei Chen and Suh Yin Lee. On-line handwritten chinese character recognition via a fuzzy attribute representation. *Image and Vision Computing*, 12(10):669–681, December 1994.

[DB73] K. A. Deffenbacher and E. L. Brown. Memory and cognition: An information processing model of man. *Theory and Decision*, 4:141–178, 1973.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley, 1973.

[EB90] D.G. Elliman and R. N. Banks. Shift invariant neural net for machine vision. *IEE Proceedings*, 137(3):183–187, 1990. Part 1.

[EDRK90] Sherif Sami El-Dabi, Refat Ramsis, and Aladin Kamel. Arabic character recognition system: A statistical approach for recognizing cursive typewritten text. *Pattern Recognition*, 23(5):485–495, 1990.

[ESET90] T. S. El-Sheikh and S. G. El-Taweel. Real-time arabic handwritten character recognition. *Pattern Recognition*, 23(12):1323–1332, 1990.

[EWS89] Mohamed S. El-Wakil and Amin A. Shoukry. On-line recognition of handwritten isolated arabic characters. *Pattern Recognition*, 22(2):97–105, 1989.

[FI93] Kunihiko Fukushima and Taro Imagawa. Recognition and segmentation of connected characters with selective attention. *Neural Networks*, 6(1):33–41, 1993.

[FM82] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982.

[FMi83] Kunihiko Fukushima, Sei Miyake, and Takauti ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions On Systems, Man, and Cybernetics*, SMC-13(5):826–834, 1983.

[FNK92] Hiromichi Fujisawa, Yasuaki Nakano, and Kiyomichi Kurino. Segmentation methods for character recognition: From segmentation to document structure analysis. *Proceedings of the IEEE*, 80(7):1079–1091, 1992.

[FOH94] Kunihiko Fukushima, Masato Okada, and Kazuhito Hiroshige. Neocognitron with dual C-cell layers. *Neural Networks*, 7(1):41–47, 1994.

[Fu74] K. S. Fu. *Syntactic Matters in Pattern Recognition.* Academic Press, 1974.

[Fu82] King Sun Fu. *Syntactic Pattern Recognition and Applications.* Englewood Cliffs, N.J., 1982.

[Fu86] King-Sun Fu. Syntactic pattern recognition. In Tzay Y. Young and King-Sun Fu, editors, *Handbook of Pattern Recognition and Image Processing*, chapter 4, pages 85–118. Academic Press, San Diego, 1986.

[Fuk88a] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

[Fuk88b] Kunihiko Fukushima. A neural network for visual pattern recognition. *COMPUTER*, pages 65–75, 1988.

[Fuk89] Kunihiko Fukushima. Analysis of the process of visual pattern recognition by the neocognitron. *Neural Networks*, 2:413–420, 1989.

[FW91] Kunihiko Fukushima and Nobuaki Wake. Handwritten alphanumeric character recognition by the neocognitron. *IEEE Transactions on Neural Networks*, 2(3):355–365, 1991.

[FW92] Kunihiko Fukushima and Nobuaki Wake. Improved neocognitron with bend -detecting cells. *International Joint Conference on Neural Network*, pages 190–195, 1992.

[Gib66] James J. Gibson. *The Senses Considered as Perceptual Systems.* Houghton Mifflin Company, Boston, 1966.

[Gib69] Eleanor J. Gibson. *Principles of Perceptual Learning and Development.* Appleton-Century-Crofts, New York, 1969.

[Gre74] R. L. Gregory. *Concepts and Mechanisms of Perception.* Gerald Duckworth, Gloucester and London, 1974.

[GU89] H. Goraine and M. J. Usher. Recognition of typewritten arabic characters in different fonts. In *IEE Colloquium on Character Recognition and Applications*, pages 9/1 - 9/5, London, October 1989.

[HC93] Keun-Rong Hsieh and Wen-Tsuen Chen. A neural network model which combines unsupervised and supervised learning. *IEEE Transactions on Neural Networks*, 4(2):357–360, March 1993.

[HK94] Basit Hussain and M. R. Kabuka. A novel feature recognition neural network and its application to character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):98–106, January 1994.

[Hoc68] J. Hochberg. In the mind's eye. In R. N. Haber, editor, *Contemporary Theory and Research in Visual Perception*, pages 309–331. Holt, Rinehart and Winston, New York, 1968.

[Hu62] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Transaction on Information Theory*, IT-8:179–187, 1962.

[Hub72] David H. Hubel. The visual cortex. In Richard Held and Whitman Richard, editors, *Perception: Mechanisms and Models*. W. H. Freeman and Company, San Francisco, 1972.

[HW68] David H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

[Jam91] Kamal M. Jambi. Arabic character recognition: Many approaches and one decade. *The Arabian Journal for Science and Engineering*, 16(4B):501–509, October 1991.

[Jul91] Bela Julesz. Early vision and focal attention. *Reviews of Modern Physics*, 63(3):735–772, July 1991.

[KPB87] Simon Kahan, Theo Pvaldis, and Henry Baird. On the recognition of printed characters of any font and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(2):274–287, 1987.

[Kuf53] Stephen W. Kuffler. Discharge patterns and functional organization of mammalian retina. *Journal of Neurophysiology*, 16(1):37–68, January 1953.

[LLS92] Louisa Lam, Seong-Whan Lee, and Ching Y. Suen. Thinning methodologies- a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, September 1992.

[Mah94] Sabri A. Mahmoud. Arabic character recognition using fourier descriptors and character contour encoding. *Pattern Recognition*, 27(0):1–10, March 1994.

[Mar73] E. Marg. Recording from single cells in the human visual cortex. *Handbook of Sensory Physiology*, VII/3B:441–450, 1973.

[Mar82] David Marr. *Vision*. Freeman, San Francisco, 1982.

[Mar92] Volker Margner. SARAT - a system for the recognition of arabic printed text. In *Proceedings of the 11th IAPR International Conference on Patern Recognition*, pages 561–564, September 1992.

[Mic72] Charles R. Michael. Retinal processing of visual images. In Richard Held and Whitman Richard, editors, *Perception: Mechanisms and Models.* W. H. Freeman and Company, San Francisco, 1972.

[Nei76] U. Neisser. *Cognition and Reality.* W. H. Freeman, San Francisco, 1976.

[NS84] Nabil Jean Naccache and Rajjan Shinghal. SPTA: A proposed algorithm for thinning binary patterns. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(3):409–418, May 1984.

[NST87] Adnan Nouh, Abobakr Sultan, and Roshdi Tolba. On feature extraction and selection for arabic character recognition. *Arabian Gulf Journal for Scientific Research*, 2(1):329–347, 1987.

[O'G90] Lawrence O'Gorman. k x k thinning. *Computer Vision, Graphics, and Image Processing*, 51:195–215, 1990.

[Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histogram. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-19(1):62–66, January 1979.

[Par91] J. R. Parker. Gray level thresholding in badly illuminated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):813–819, 1991.

[Ros59] R. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, 1959.

[Sch92] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* John Wiley and Sons, Inc., New York, 1992.

[Sri92] Sargur N. Srihari. High performance reading machines. *Proceedings of the IEEE*, 80(7):1120–1132, July 1992.

[WE92] Brian A. White and Mohammed I. Elmasry. The digi-neocognitron: A digital neocognitron neural network model for vlsi. *IEEE Transactions on Neural Networks*, 3(1):73–85, January 1992.

[Woz88] P. D. Wozerman. *Neural Computing: Theory and Practice.* 1988.

[YO93] Cem Yuceer and Kemal Oflazer. A rotation, scaling, and translation invariant pattern recognition system. *Pattern Recognition*, 26(5):687–710, 1993.

[ZG91] Xiaohua Zhou and Richard Gordon. Generation of noise in binary images. *CVGIP: Graphical Models and Image Processing*, 53(5):476–478, September 1991.

Vita

- Tambi Mohammed Sharief Baik
- Born in 1969 Damascus, Syria.
- Received the Bachelor of Science degree in Computer Science in 1992 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Received the Master of Science degree in Computer Science in 1995 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.